The Waite Group

PLUME
WAITE

GAMES & UTILITIES FOR THE TRS-80® MODEL 100

A LIBRARY OF
PROFESSIONAL-QUALITY PROGRAMS

- Exciting, arcade-style action games
- Practical, powerful utilities
- Short, easy-to-enter programs
- Complete explanations of program operation

by
Ron Karr, Steven Olsen,
and Robert Lafore

Another book by the bestselling authors of *ASSEMBLY LANGUAGE PRIMER FOR THE IBM® PC & XT* and *BLUE-BOOK OF ASSEMBLY ROUTINES FOR THE IBM® PC & XT*. . . . And rave reviews for The Waite Group:

"An outstanding example of how to write a technical book for the beginner . . . refreshingly enjoyable . . . accurate, readable, understandable, and indispensable. Don't stay home without it."

— Ken Barber, reviewing *CP/M Primer*, in *Microcomputing*

"Mitch Waite . . . has left a distinctive contribution to the literature of computer graphics . . . seeing it here is like understanding it for the first time."

— *Computer Graphics Primer*, reviewed in *Computer Graphics World*

"It's hard to imagine that a field only a decade old already has a classic, but Waite's book is just that."

— Tony Dirksen, reviewing *Computer Graphics Primer* in *Interface Age*

". . . an outstanding reference work, aside from its obvious benefit as an instructional text . . . superb writing style . . ."

— Chuck Dougherty, reviewing *Soul of CP/M* in *Cider*

"The demystification of a complex technological development . . . rating: 100."

— *8086/8088 16-bit Microprocessor Primer*, reviewed in *The Reader's Guide to Microcomputer Books*

"[This book] can have anybody writing and understanding assembly language programs within one hour."

— Alan Neibauer, reviewing *Soul of CP/M*, in *80 Microcomputing*

"All-inclusive, beautifully organized, easy-to-use reference that helps you wrest order from chaos . . ."

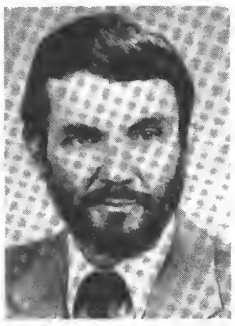
— *CP/M Bible*, reviewed in *Byte Book Club Bulletin*

Ron Karr

Ron Karr is a games designer and programmer in San Rafael, California. He is an expert bridge and backgammon player, and for many years made his living as a professional blackjack player, or "card counter," in Las Vegas and Reno. He is currently working with software games designer Ken Uston, writing games software for the ATARI® and Commodore 64. He is experienced in several computer languages, including BASIC and 6502 assembly language. Ron's non-game interests include philosophy, running, and the piano.

Steven Olsen

Steven Olsen is a computer programmer/analyst with the U.S. Government. A graduate of the University of South Florida, he has been involved with computers and digital electronics for the last 13 years and is also a consultant in custom hardware and software design. Mr. Olsen has experience in teaching computer use, programming, and communications, and an extensive background in programming and database development.



Robert Lafore

Robert Lafore is managing editor of the Waite Group, a company which produces computer books in Sausalito, California. Mr. Lafore has worked with computers since 1965, when he first learned assembly language on the DEC PDP-5. He holds degrees in Mathematics and Electrical Engineering, and is the author of *Assembly Language Primer for the IBM® PC and XT*, and co-author of *Soul of CP/M*, an assembly language book for CP/M systems. Mr. Lafore founded Interactive Fiction, a computer game company; and has also been a petroleum engineer in Southeast Asia, a novelist, a newspaper columnist, a systems engineer for the University of California's Lawrence Berkeley Laboratory, and has sailed his own boat to the South Pacific.

GAMES & UTILITIES FOR THE TRS-80® MODEL 100

by Ron Karr,
Steven Olsen,
and Robert Lafore

A Plume/Waite Book
New American Library
New York and Scarborough, Ontario

Several trademarks and/or service marks appear in this book. The companies listed below are the owners of the trademarks and/or service marks following their names.

Microsoft: MBASIC
MicroPro International Corporation: WordStar
Tandy Corporation: TRS-80 Model 100 Portable Computer


PLUME TRADEMARK REG. U.S. PAT. OFF. AND FOREIGN COUNTRIES
REGISTERED TRADEMARK — MARCA REGISTRADA
HECHO EN WESTFORD, MASS., U.S.A.

SIGNET, SIGNET CLASSIC, MENTOR, PLUME, MERIDIAN and NAL BOOKS are published *in the United States* by New American Library, 1633 Broadway, New York, New York 10019, *in Canada* by The New American Library of Canada Limited, 81 Mack Avenue, Scarborough, Ontario M1L 1M8

Cover design by Michael Manwaring
Illustrations by Stuart Bradford
Technical illustrations by Karen and Winston Sin
Typography by Walker Graphics

First Printing, November, 1984

1 2 3 4 5 6 7 8 9

PRINTED IN THE UNITED STATES OF AMERICA

Contents

Preface

This book is the result of a collaborative effort.

Ron Karr wrote most of the game programs, specifically, Toxic Raiders, Interjerk Invaders, High-Tech-Tic-Tac-Toe, Casino Blackjack, Portable Piano, Noterony, Micro-Rembrandt, Mega-Rembrandt, Conway's Life, Dueling Digits, Typerony, Mathomania, Electric Hangman, and Alphabet Soup.

Steven Olsen wrote the majority of the utility programs, including Easycalc, Mr. Metric, Hexidec, Perpetual Calendar, Days Between, Superwatch, Easy Sort, Formatter, Statistics, Probability, and Primes. He also wrote the Star Stomp game.

Robert Lafore wrote the utilities Sci-Calc, Listfile, Trace, and File Length. In addition, he wrote many of the program descriptions, rewrote others, and is responsible for the final form of the text.

Mitchell Waite modified and improved a number of the game programs.

Bob Petersen wrote the utilities Any Sort and Word Count.

Kim House wrote many of the utility program descriptions, rendered invaluable assistance in pulling together the final manuscript, and generally burned the midnight oil to ensure that this book was as good as it could be.

Introduction

You are the proud owner of a Radio Shack TRS-80® Model 100 Portable Computer. You have discovered that, although you can take it anywhere in a small briefcase, it still has the power of many larger desktop computers. You've learned something about how to use it, perhaps by reading *Introducing the TRS-80® Model 100* by Diane Burns and Sharyn Venit (New York: Plume/Waite, New American Library, 1984), another book in this same series. You can type letters, arrange your schedule, and talk to other computers over the phone lines, all using the Model 100's built-in programs.

The question is, what next?

What's in This Book?

This book is designed to answer the question, "What can I do with my new computer?" In it you will find a variety of programs covering the entire spectrum from frivolous and exciting games to serious calculation and file sorting programs. A glance through the Table of Contents should give you an idea of the kinds of programs included. To use these programs you don't need to know anything about programming — anyone can type them in and use them. We show you exactly how in the next chapter.

Do you have some free time on the New York to London flight? Try one of the arcade games. You may get so excited you'll be glad you kept your seat belt on. Are you a journalist, being paid by the word? Don't let *MicroCraze* magazine cheat you — count the number of words in your article with the Wordcount program. Do you need to do some quick or even complex calculations? This book has two calculator programs: your choice of a simple five-function calculator or a sophisticated scientific RPN calculator. Are your children struggling with their multiplication tables? The Dueling Digits program teaches multiplication, and is so much fun the kids won't notice how fast they're learning.

Easy Access

One of the really great things about the Model 100 is that once you've entered a program into memory, you can keep it there as long as you want, unlike other computers where the programs are erased whenever you turn the computer off. This means you can type in a program once, and then access it instantly whenever you like, simply by positioning the cursor on the program name in the main menu and pressing the ENTER key.

For instance, you might want to keep a calculator, a few of your favorite games, the metric conversion program, a file sorting program, and the perpetual calendar program all in memory at the same time. With each program instantly accessible, you can switch back and forth from one to another whenever you want.

Short and Sweet

No one likes to type in long programs, so one of our main aims in creating the programs for this book was to keep them as short and efficient as possible. There's something very discouraging about the prospect of typing in a five-page program listing. Such programs may look impressive in a book or magazine, but the chances are they'll never really be used. Also, it's easier to avoid typing errors in short programs, and if you do make one (it happens to everyone occasionally!) it's much easier to find.

The programs in this book are often less than a page long; some are only a few dozen lines. And yet they are very powerful. Sophisticated programming techniques and the inherent power of the Microsoft® BASIC built into the Model 100 mean that even short programs can produce exciting and versatile results.

Programmer's Delight

Although you don't need to be a programmer to use this book, the programs offer many benefits to both the beginning BASIC programmer and the expert.

The beginning programmer will find the programs a valuable learning device. The operation of each program is explained in detail in the sections called "How the Program Works". This nuts-and-bolts explanation follows each program listing. By reading these sections and studying the listing, the novice (or intermediate) programmer will gain insight into programming technique and learn how to access the Model 100's special features.

The expert programmer will find the programs to be convenient jumping-off points. Since the programs are short, clearly written, and well-documented, they can be easily modified to meet the specific needs of individual users (in some cases we even suggest ways this might be done). Or the programs can serve as models for longer, more ambitious programs.

Typing in and Running the Programs

In this chapter we'll briefly review how to type in the programs. This process is easy. Simply follow the few short steps explained below, and you'll be running the programs in no time. We're not going to try to give a complete course in BASIC here — to learn more about the language, read *Mastering BASIC on the TRS-80® Model 100,* by Bernd Enders (New York: Plume/Waite, New American Library, 1984), another book in this series. However, we will tell you everything you need to know to type in the programs in this book, save them to memory, and run them whenever you want. If you already know BASIC you can skim over this chapter, or skip it altogether.

Getting into BASIC

Let's assume that you've just turned the computer on and are looking at the "main menu", the display showing all the filenames, with the date and time at the top of the screen. Since the programs are written in BASIC, you need to put the computer into BASIC programming mode to type them in. To do this, you use the cursor keys to position the cursor (the black rectangle) over the word BASIC in the upper left-hand corner of the screen. When you've done this, press the ENTER key. The computer will clear the screen and display

```
TRS-80 Model 100 Software
Copr. 1983 Microsoft
23197 Bytes free
Ok                          ← BASIC's prompt
▒                           ← Blinking black rectangular cursor
```

Whenever you see the BASIC prompt "Ok", you know that BASIC is waiting for you to type something in. The blinking cursor shows you *where* what you type is going to go.

Typing in a Sample Program

Let's type in a very short sample program to demonstrate the steps involved. This program will print the phrase "Good Morning" on the screen.

```
10 PRINT "Good Morning"
20 END
```

Type it in just as it's shown above.

Running the Program

To run a program which you've typed in, you can either enter the word "RUN", or you can press function key (F4). Either way, the program will start running, and in the case of our sample program, the message "Good Morning" will be printed out on the screen.

Typing Details

There are some things to keep in mind when typing in programs.

Lower Case

It's all right (though not necessary) to use lower case letters when typing in BASIC keywords such as PRINT and END. BASIC will automatically convert these letters to upper case. However, anything in quotes or following the word DATA should be typed in exactly as shown in the program listings.

Space after Line Number

The numbers on the left in our sample program listing, "10" and "20", are called "line numbers." Don't forget the space following these numbers and before the rest of the program line.

Line Lengths

A BASIC program line starts with a line number and ends when you press (ENTER). This is true no matter how many characters are in it or how many lines it takes up on the screen or in the book.

You can only fit forty characters in the width of the Model 100 screen. For clarity, many lines in the programs in this book are longer than 40 characters. How is this handled?

When you come to the end of the Model 100 screen and still have characters left to type, the important thing to remember is to *just keep typing!* Don't try to press the ENTER key to move down to the next line. If you keep typing, the cursor will drop down automatically to the next line. Don't worry if a word is broken in the middle; BASIC doesn't mind where a program line is broken on the screen.

Some program lines are too long to fit on one line in the book. If you see a line in a listing in the book that *doesn't* start with a line number, it is part of the preceding BASIC program line. Some BASIC lines take up to three lines in the book, which may be equivalent to five or more lines on the Model 100 screen. There's no problem as long as you remember not to press ENTER until the entire BASIC program line is complete, no matter how many book lines or screen lines it occupies.

At the End of Each Line

The ENTER which you press at the end of each program line tells BASIC to "digest" the line; that is, incorporate it into the program at the line number shown.

Potential Mistakes

Be careful not to confuse the number "1" and the letter "l", the number "0" and the letter "O", or the number "6" and the letter "G".

Listing and Editing the Program

Once the program is typed in, you can "list" or display it by typing the word LIST (either upper or lower case) and pressing ENTER. It's a good idea to list the program this way after you have typed it in, so that you can compare it to the version in the book and make sure you haven't made any mistakes.

Here's how this process looks with our sample program:

```
Ok
list                              ← Enter this to list the program
10 PRINT"Good Morning"            ← Listing starts here
20 END
```

On a long program you will need to stop the listing before it displays too many lines. Do this by pressing the (PAUSE) key. To start the listing scrolling again, press the (PAUSE) key again. You can start and stop the display this way until you have seen the entire listing.

Correcting Mistakes

If you find a mistake in a program line, you'll need to correct it. There are two ways to do this. The first is to retype the entire line. Start over with the line number and retype it. The old line with the same line number will be overwritten and the new line substituted in its place.

The second way to correct program lines is to use the EDIT command. The advantage here is that you can make corrections without retyping the entire line. If the line is long and you've only made one small error, you'll find this very useful.

To use the EDIT command, type the word "EDIT" followed by the line number of the line you want to correct. For instance, suppose you wrote "Mourning" in line 10 instead of "Morning".

```
Ok
EDIT 10                ← Type this to edit line 10
```

The screen will clear, and line 10 will be displayed at the top of the screen. You can now edit the line using the following commands:

Move the cursor to any part of the line with the cursor control arrows (←) and (→).

Insert characters simply by positioning the cursor at the correct part of the line and typing them in.

Delete characters by pressing (DEL/BKSP), which deletes the character to the left of the cursor; or by holding down (SHIFT) and pressing (DEL/BKSP), which deletes the character directly under the cursor.

When you've finished a correction press function key (F8) to get out of editing mode and back to the BASIC "Ok" prompt.

Saving the Program in Memory.

Once the program is typed in and you know it works, you can save it in RAM (Random Access Memory). This allows you to run it whenever you want, directly from the main menu.

To save the program you've just typed in, type the word SAVE, then quotes ("), then the name of the program, then quotes again ("). Then press the ENTER key. The name can be anything you wish; however, it must start with a letter and be less than six characters long. You can type it in upper or lower case; BASIC will convert it to upper case.

Let's say we're going to call our example program "GOOD". Here's what you type:

```
Ok
SAVE"GOOD"
Ok
```

Now if you want to go back to the main menu, press function key F8. You'll find the program listed there as "GOOD.BA". BASIC always adds the file extension BA to all BASIC programs.

Erasing a Program

If you followed the steps above, you will now have a short program called "GOOD" in memory. You probably don't really want it there, so you'll need to know how to delete it. Files are deleted from within BASIC, so get into BASIC as described above. When the "Ok" prompt appears, type:

```
Ok
KILL"GOOD.BA"        ← Enter this to delete file
```

Note that you *must* include the file extension BA when using the KILL command (unlike the "SAVE" command).

Carry On

With this background you should have no trouble typing in the programs, saving them in RAM, and executing them whenever you like. Try it! We think you'll enjoy yourself.

3

Arcade Games

RAIDER

Toxic Raiders

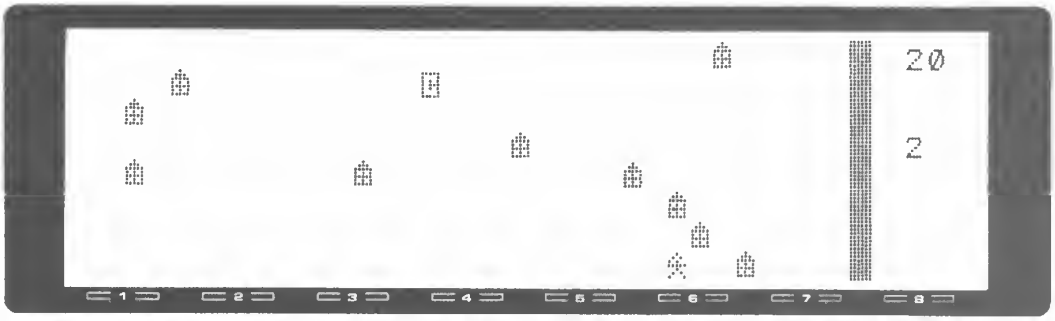
Fast Paced Cliffhanger

Toxic Raiders is an exciting arcade-style game that will have you on the edge of your chair. It requires steady nerves and quick reflexes. In the game you are represented by a small humanlike figure (the hero), which you can move around the screen using the keyboard keys. You have two goals — to attack and destroy a number of enemy factories creating toxic waste, and, at the same time, to avoid being gobbled up by an enormous letter "C", which stands for Corporate Greed.

You control your man with four keyboard keys:

Y = up
H = down
G = left
J = right

The use of these keys may seem somewhat arbitrary at first, but when you try it, you'll find they fit more naturally under your fingers than the regular cursor keys. Position your index finger on the "G", your middle finger on the "H", and your ring finger on the "J". Now you can move your middle finger back and forth between the "Y" and the "H" to control vertical motion, and use your index and ring fingers to control horizontal motion.



When you start the game, you'll be asked whether you want the hard or the easy version. Reply by typing "H" for hard or "E" for easy. You'll be well-advised to start with the easy version. The hard version requires both practice and pantherlike reflexes.

When you start the game, the big "C" heads right for you. You need to avoid it, and, at the same time, gobble up as many factories as you can. The "C" is very fast and completely relentless since it uses a computer-controlled homing device. However, you do have one advantage over it (besides your native intelligence and general goodness): you can "wrap around" the screen, and it can't. That is, when you come to the side of the screen, you immediately appear on the other side; whereas the big "C" must turn around and head all the way back across the screen to catch you.

You start out with four turns or "lives". Each time the big "C" catches you, you lose a turn. When you've lost all four turns, the game is over. However, if you can destroy ten factories without being caught by the big "C", you get an extra turn. Each time you destroy a factory, you get ten points. The object of the game is to amass as many points as possible.

Join the fight against pollution! Join the Toxic Raiders!

Program Listing

```
10 '
20 '   RAIDER
30 '
40 ' Initialize game
50 '
60 DEFINT A-Z
70 CLS: PRINT "Enter h for hard, e for easy: ";
80 C$=INKEY$: IF C$="" THEN 80
90 IF C$="h" OR C$="H" THEN EA=0
100 IF C$="e" OR C$="E" THEN EA=-1
110 DIM D$(319)
120 V=VAL(RIGHT$(TIME$,2))
130 FOR I=0 TO V:R=RND(1):NEXT I
140 H$=CHR$(148):M$=CHR$(171):S=0:MM=4
150 A$=" ":B$=" ":TT=0
160 '
170 ' Print Starting Layout
180 '
190 CLS:FOR I=34 TO 314 STEP 40:PRINT@I,CHR$(239);:NEXT I
200 PRINT@35,S;:PRINT@155,MM;
210 Y=4:X=33:H=193
220 PRINT@H,H$;:X1=X:Y1=Y
230 J=3:I=0:M=120
240 PRINT@120,M$;:J1=J:I1=I
250 '
260 ' Print Targets
270 '
280 FOR K=0 TO 319:D$(K)=" ":NEXT K
290 FOR K=1 TO 10
300 U=INT(RND(1)*32+1):V=INT(RND(1)*8)
310 PD=U+40*V
320 IF D$(PD)<>" " THEN 300
330 D$(PD)=CHR$(134):PRINT@PD,D$(PD);
340 NEXT K
350 B$=INKEY$:B$=INKEY$:B$=INKEY$:B$=INKEY$
360 '
370 ' Move Hero
380 '
390 B$=INKEY$:IF B$="" THEN B$=A$
400 IF B$="j" THEN X=X+1:IF X>33 THEN X=0
410 IF B$="g" THEN X=X-1:IF X<0 THEN X=33
420 IF B$="h" THEN Y=Y+1:IF Y>7 THEN Y=0
430 IF B$="y" THEN Y=Y-1:IF Y<0 THEN Y=7
440 H=X+40*Y:H1=X1+40*Y1
```

```
450 PRINT @H1," ";:PRINT @H,H$;
460 X1=X:Y1=Y:A$=B$
470 IF D$(H)=" " THEN 510
480 SOUND 16000,2:S=S+10:D$(H)=" ":PRINT@35,S;:TT=TT+1
490 IF TT=10 THEN 150 ELSE 580
500 '
510 ' Move Chaser
520 '
530 ' If "hard" then move chaser, else move every other time
540 '
550 IF NOT(EA) THEN 580
560 F=NOT(F):IF F=-1 THEN 390
570 '
580 IF X=I THEN 600
590 IF X>I THEN I=I+1 ELSE I=I-1
600 IF Y=J THEN 620
610 IF Y>J THEN J=J+1 ELSE J=J-1
620 M=I+40*J:M1=I1+40*J1
630 PRINT @M,M$;:PRINT@M1,D$(M1);
640 J1=J:I1=I
650 IF M<>H THEN 390
660 SOUND 8000,4:MM=MM-1
670 IF MM=0 THEN 730
680 PRINT@35,S;:PRINT@155,MM;
690 GOTO 150
700 '
710 ' End of Game
720 '
730 PRINT@95,"GAME OVER";
740 PRINT@155,MM
750 PRINT@175,"Score: ";S;
760 R$=INKEY$:R$=INKEY$
770 R$=INKEY$:IF R$="" THEN 770
780 GOTO 120
```

How the Program Works

The first sections of the program, lines 10 to 240, set the initial values of the variables, generate random numbers to be used later, draw the initial screen, and request the difficulty level.

Lines 390 to 650 form a loop that controls one complete "play" in the game.

The routines in lines 280 to 350 generate ten random locations on the screen where the targets will be placed. There are 320 such possible locations, represented by the array D$. If an element of the array contains the character CHR$(134), then there is a factory in that location; otherwise the array element contains a blank (" ").

Lines 390 to 490 read the keyboard input (which letter key you've pressed) and move the hero in the appropriate direction. The variables X and Y represent the coordinates of the hero, which are translated into the print position on the screen in line 440. Line 470 then checks to see whether there is a target at that position. If there is, a sound is made and ten points are added to the score.

In lines 580 to 610, the coordinates of the hero and the big "C" are compared, and the "C" is moved one position closer to the hero. If the coordinates of the "C" and the hero are the same, then the turn will be over. This is checked in line 650. If all four turns are used up, which is checked in line 670, then the game is over and lines 730 to 780 are executed. If turns remain, control branches back to line 150.

VADERS

Interjerk Invaders

Save the World, If You Can

The world is being taken over by hordes of aggressive, soulless, materialistic numbers! These inhuman digits don't care anything about you, or your hopes and dreams. They're infiltrating everything that we humans hold dear, and it's happening right now, before your very eyes! Can nothing be done? Actually, yes. In this spine-chilling game, your mission is to destroy a horde of ravenous numbers as they march inexorably across the screen. Can you do it? Will your bravery, skill, and lightning reflexes make the world safe once again for those of us who have not yet been digitized?

A "digit horde" consists of 49 numbers, arranged in seven rows of seven numbers each. Because numbers are so regimented, each row contains one each of exactly the same numbers, the digits 1 through 7. The digit horde starts out on the left side of the screen and attempts to advance across the screen to the right. As it does this, it sends out fast "scouts" to reconnoiter and penetrate your position. Your mission is to stop them. You can do this in two ways. You (represented by a small, lonely human figure) can either try to block the numbers by throwing yourself in their path, kamikaze-style, or you can vaporize them by firing a small circular blob of poetry at them — the interjerk digits hate poetry.

You block the invaders by moving your human figure up and down into the path of the oncoming scouts. This is accomplished by pressing the letters

"u" for up and "j" (or any other key) for down. You shoot the interjerk digits by pressing "f". Once you've fired a shot, you can't fire again until the first shot has reached its target, so you must "aim" carefully. Your shots travel toward the invaders on the same row as your figure.

When you score a hit on an interjerk invader, either by blocking it or hitting it, you get the number of points which it represented added to your score; thus, if you hit a "7", your score is increased by 7 and so on. However, each time an invader gets past you to the right-hand edge of the screen, his number of points is *subtracted* from your score.

Each complete game consists of three "waves" of number invaders. Each wave is tougher than the last: it starts closer to the right side of the screen, but it has higher numbers, so the stakes are raised as the game continues.

Program Listing

```
10 '
20 ' VADERS
30 '
40 DEFINT A-Z:DIM A$(6)
50 S=0:M$=RIGHT$(TIME$,2)
60 FOR I=0 TO VAL(M$):J=RND(1):NEXT I
70 FOR Z=0 TO 2
80 '
90 ' New Screen
100 '
110 P=38:H$=CHR$(148):TT=0:F=0
120 FOR I=0 TO 6
130 A$(I)=MID$("987654321",(3-Z),7)
140 NEXT I
150 CLS:PRINT@P,H$;:PRINT@280,"Score:    ";S;:
160 '
170 ' Start Loop to Print Rows of Invaders
180 '
190 FOR J=Z TO 49
200 IF J>33 THEN J=J-1
210 IF TT>48 THEN 690
220 FOR I=0 TO 6
230 PRINT@(I*40+J),STRING$((Z+1)," ");A$(I);
240 NEXT I
250 IF F=1 THEN RP=RP+1:IF FP=RP THEN GOSUB 740
260 '
270 ' Pick Invader to Move
280 '
290 I=INT(RND(1)*7)
300 B=LEN(A$(I)):IF B<=0 THEN 290
310 C$=RIGHT$(A$(I),1):A$(I)=LEFT$(A$(I),B-1)
320 '
330 ' Start Loop to Move Invader
340 '
350 FOR N=I*40+J+B+Z+1 TO I*40 +38
360 PRINT @N,C$;:PRINT@(N-1)," ";
370 IF N=FP THEN 550
380 K$=INKEY$:IF K$="" THEN 460
390 IF K$="f" OR K$="F" THEN 510
400 PRINT @P," "
410 IF K$="u" OR K$="U" THEN 440
420 P=P+40:IF P>278 THEN P=278
430 GOTO 450
440 P=P-40:IF P<38 THEN P=38
450 PRINT @P,H$
```

```
460 IF F<>1 THEN ST=15:GOTO 560
470 IF (FP MOD 40)<(J+Z) THEN F=0:GOTO 570
480 '
490 ' Fire a Shot
500 '
510 IF F=0 THEN F=1:FP=P-1:PRINT@FP,"o";:
    K=INT(FP/40):RP=40*K+J+Z+LEN(A$(K))
520 FP=FP-1:PRINT@FP,"o";:PRINT@(FP+1)," ";
530 IF FP=RP THEN GOSUB 740:GOTO 570
540 IF FP<>N THEN ST=1:GOTO 560
550 SOUND 500,2:S=S+VAL(C$):F=0:PRINT@N," ";:FP=999:GOTO 640
560 FOR T=0 TO ST:NEXT T
570 NEXT N
580 '
590 ' End Move Invader--Check Whether it Landed or Was Blocked
600 '
610 IF (N-1)<>P THEN 630
620 PRINT@P,H$;:SOUND 2000,2:S=S+VAL(C$):GOTO 640
630 S=S-VAL(C$):SOUND 1000,1:PRINT@(N-1)," ";
640 PRINT @290,S;:TT=TT+1
650 NEXT J
660 '
670 ' End Loop to Print Rows
680 '
690 NEXT Z
700 Q$=INKEY$:IF Q$="r" OR Q$="R" THEN 50 ELSE GOTO 700
710 '
720 ' Hit Invader Routine
730 '
740 F=0:PRINT@RP," ";:FP=999
750 BR=LEN(A$(K)):IF BR=0 THEN RETURN
760 R$=RIGHT$(A$(K),1):S=S+VAL(R$):A$(K)=LEFT$(A$(K),BR-1)
770 SOUND 500,2:PRINT@290,S;:TT=TT+1:RETURN
```

How the Program Works

Lines 10 to 70 initialize diverse variables. They also "seed" the random number generator. "Seeding" means to extract a random amount of random numbers from the generator function so that the random numbers used by the program won't always start with the same sequence.

The variable Z in line 70 tells you which of the three "waves" or phases of the game is being executed. It's used in a FOR...NEXT loop which goes from line 70 all the way to 690. The variable Z is then used in line 130 to specify the exact configuration of the line of invaders for the three different waves. The first wave consists of the digits 1 to 7, the second 2 to 8, and the third 3 to 9.

The string array A$ holds the row of number invaders. At the beginning of the game, each row is the string "7654321", but, as noted above, this changes for each wave. The section of code from line 90 to line 150 is responsible for setting up each element of the array with this string.

The variable J indicates how far across the screen, from left to right, the invaders have progressed. It is used in another large loop, from 190 to 650. Each time this loop is executed J is incremented and the invaders all move one space closer to the right, toward the last non-numeric being on earth.

In lines 290 to 310 a random row is picked, and the far right invader, represented by variable C$, is launched as a fast-moving "scout". Its position, represented by N, is incremented in the loop between lines 350 and 570. This continues until it reaches the right edge of the screen, or is shot or blocked.

Line 610 checks whether your man is at the same location as the scout. If so, you have successfully blocked the scout, and your score is increased accordingly.

The program checks to see if a shot was fired (the "f" key pressed) in line 390. If so, control jumps to line 510. The variable F, when it is set to 1, indicates that a shot is already in progress. In this case no further shots are allowed, and the position of the current shot is updated in line 520. However, if F=0, the shot, represented by the letter "o", is started at print position FP. RP is the print position of the rightmost invader in the row that is being fired at. When FP equals RP, as determined in line 530, the program jumps to the routine at 740. Here both the shot and the invader are erased, the score is updated, and a "beep" is sounded to indicate a hit. F is reset to 0, and control returns to line 570 to read the keyboard again.

4

Table Games

TICTAC

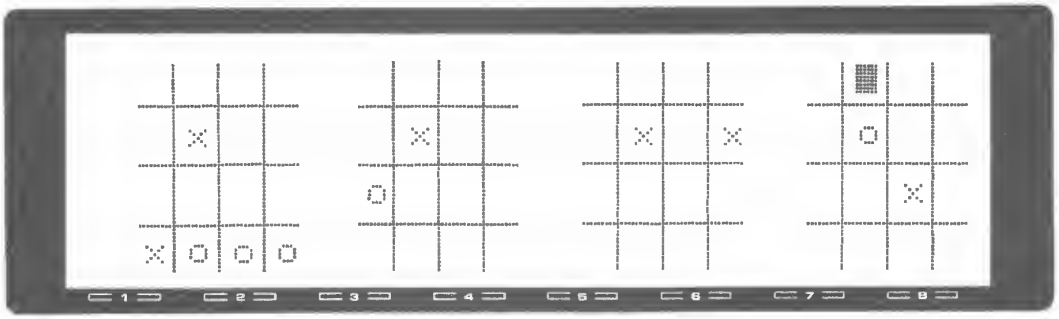
High-Tech-Tic-Tac-Toe

Three-Dimensional Version

Remember the classic Tic-Tac-Toe game with the zeros and crosses you used to play as a kid? Here's a modernized, computerized, three-dimensional, four-in-a-row version of the old game. It's still a two-player game, but this version adds a whole new level of perception and strategy. It's so advanced you might find Mr. Spock playing it with Captain Kirk, when they tire of three-dimensional chess.

This game uses a four-by-four-by-four matrix instead of the old three-by-three. That is, it has four four-by-four planes, stacked on top of each other, in effect, creating a four-by-four-by-four cube. For simplicity, the screen shows each of the four levels as a separate picture, as shown in the illustration. When you visualize the game, you'll need to think of these four planes stacked on top of each other. It doesn't really matter whether you think of the left-hand plane or the right-hand plane as being on top, as long as you visualize the *opposite* plane on the bottom, with the two middle ones between.

Just as in the classic Tic-Tac-Toe game the players alternate turns. To make a move, you use the four cursor keys, ↑, ↓, ←, →, to move a "block cursor" (a small rectangle) to the square of your choice. Then, depending on whether you are the first player or the second, you type either



an "x" or an "o" (that's the letter "o"). Make sure you don't have the CAPS LOCK key toggled to caps, since only lower case "x" and "o" will work. If you make a mistake, you can delete any entry by placing a space at the offending location.

As you play, the program keeps track of where your "x"s and "o"s are. As soon as either player gets four in a row, either on one plane or with one "x" or "o" on each plane, the program announces the fact with a flashing message and a spirited trumpet fanfare. Also, the four characters which constitute the four in a row are flashed on and off, making it very clear where they are. This is not a frivolous capability, for it is often difficult to see, in the maze of x's and o's spread out over three dimensions, just where the winning markers are.

Program Listing

```
10 '
20 ' TICTAC
30 '
40 DIM V(13),X$(278)
50 DATA 2,10,80,8,12,78,82,68,70,88,90,92,72
60 FOR I=1 TO 13:READ V(I):NEXT I
70 FOR I=0 TO 278 STEP 2:X$(I)=" "
80 NEXT I
90 '
100 ' Draw Grids
110 '
120 CLS
130 FOR K=9 TO 189 STEP 60
140 FOR I=K TO K+24 STEP 12
150 LINE (I,0)-(I,56)
160 NEXT I:NEXT K
170 FOR I=12 TO 44 STEP 16
180 FOR J=0 TO 180 STEP 60
190 LINE (J,I)-(J+42,I)
200 NEXT J:NEXT I
210 '
220 ' Move Cursor, Enter x's and o's
230 '
240 P=0:PRINT@0,CHR$(239)
250 A$=INKEY$:IF A$="" THEN 250
260 IF A$="x" OR A$="o" OR A$=" " THEN 310
270 P1=P+(2 AND A$=CHR$(28))
    -(2 AND A$=CHR$(29))
    +(80 AND A$=CHR$(31))
    -(80 AND A$=CHR$(30))
280 IF P1>278 OR P1<0 THEN P1=P
290 PRINT @P1,CHR$(239): PRINT@P,X$(P)
300 P=P1:GOTO 250
310 PRINT @P,A$:X$(P)=A$
320 IF A$=" " THEN 250
330 '
340 ' Check for Winner by Comparing Print Intervals
350 '
360 FOR I=1 TO 13
370 W=V(I)
380 FOR J=1 TO 3
390 Y=P+W*J
400 IF Y>278 THEN 440
410 IF X$(Y)<>A$ THEN 440
```

```
420 NEXT J
430 GOTO 550
440 FOR K=1 TO 4-J
450 Y=P-W*K
460 IF Y<0 THEN 500
470 IF X$(Y)<>A$ THEN 500
480 NEXT K
490 GOTO 550
500 NEXT I
510 GOTO 250
520 '
530 ' Indicate Winner
540 '
550 SOUND 4697,5:SOUND 3516,5:SOUND 2793,5:
    SOUND 2348,10:SOUND 2793,5:SOUND 2348,15
570 IF J=4 THEN Y=P
580 FOR S=0 TO 3
590 PRINT @133,"              "
600 PRINT@(Y+S*W),A$:NEXT S
610 FOR S=0 TO 3
620 PRINT @133,"** ";A$;" WINS **"
630 PRINT@(Y+S*W)," ":NEXT S
640 I$=INKEY$:IF I$="" THEN 580
650 GOTO 70
```

How the Program Works

The first two tasks performed by this program are fairly straightforward. After some initialization statements (which we'll get back to later), the program draws the game grid (lines 100 to 200), using LINE statements. Then the block cursor is drawn in the upper left-hand corner, and the program uses an INKEY$ statement (line 250) to see which character the user has typed in on the keyboard. If the character is an "x", an "o", or a space, then that character is printed on the display at the current cursor position. Otherwise, the character is assumed to be a cursor key, and the current print position is updated to reflect the movement of the cursor in line 270.

The big problem the program must solve is figuring out when there is a winner; that is, when four x's or four o's form a row. The solution to this problem makes use of the fact that if there are characters on the screen which constitute four-in-a-row, they will occupy positions equal distances apart in terms of the *screen print positions*. These are the positions, from 0 to 319, used in the PRINT@ statement. These print positions start at 0 in the upper left-hand character position on the screen and run across each of the eight rows, in turn, to the lower right-hand corner, position 319.

For four characters to form four-in-a-row, there must be equal intervals between them when they are printed. Thus, in the simplest case, a horizontal row on the top row of the left-hand plane would occupy print positions 0, 2, 4, and 6. These are all separated by an interval of two. More complicated four-in-a-rows, such as diagonals and rows on different planes, are separated by different intervals.

The initialization part of the program contains all these intervals in a DATA statement (line 50). These thirteen intervals are first read into an array V. The print positions occupy an array X$, which is initially filled with spaces.

Each time one of the players makes a move, the program, in lines 340 to 510, checks to see if this move makes four-in-a-row. It does this by checking the list of print positions in array X$, to see if there are four separated by one of the intervals in array V. This involves nested loops, using index I to step through the intervals in array V, and indices J and K to step through the intervals in array X$.

JACK

Casino Blackjack

The Complete Cardgame

*H*ave you have ever fantasized about becoming a big-time gambler with the knowledge and poise of James Bond? Can you picture yourself making big money at the tables? Casino Blackjack allows you to play the game of blackjack, or "21", exactly as it is dealt in the casinos of Las Vegas. This Model 100 program will teach you all you need to know to play a ruthless, high-stakes game against the experts.

Unlike many computer blackjack programs, this one handles all the intricacies of the real game, such as keeping track of your bankroll, splits, doubles, and insurance bets.

The Rules of Blackjack

As a game begins, the dealer shuffles the deck, and then deals two cards face up to the player (that's you), and two cards to himself. One of the two cards dealt to the dealer is face up, the other face down. Each card represents a certain number of points. The cards from 2 to 10 count as their face values. Aces may be counted as either 1 or 11, whichever is more advantageous for the player. Picture cards (jacks, queens, kings) count as 10. The cards 10, jack, queen, king, and ace are indicated in the program by the symbols T, J, Q, K, A.

The Object of the Game

The object of the game is to get a total — the sum of the points on all your cards — higher than the dealer's, without exceeding 21. If you succeed in this, you win the hand. If you don't — if you have fewer points than the dealer, or if you exceed 21 — you lose.

Blackjack is, of course, a gambling game, so how you are doing overall is measured in terms of your bankroll. You start off with a certain amount of money. You bet a portion of this bankroll on each hand — if you win, you get your bet back, plus an equal amount from the dealer. If you lose, you lose your bet.

The game uses a single deck of 52 cards. After each hand, the cards for the next hand are dealt from the cards remaining in the same deck, unless there are fewer than 13 cards remaining, in which case the deck is shuffled again, and the game starts over with a new deck.

Strategy

In general, the player exercises skill and judgment in blackjack by making decisions at various points in the game. However, there is one situation that requires no decision-making. If the player's first two cards are an ace and another ten-value card, he has "blackjack". This immediately pays one and one-half times the amount bet (unless the dealer also has blackjack, in which case it's a tie, and the player simply gets his bet back). The dealer can also get blackjack, in which case the player loses.

The primary place to exercise judgment is in deciding whether or not to draw another card after the first two cards are dealt. Drawing another card is called a "hit". If you have already been dealt cards which total very close to 21, such as a 9 and a 10, then chances are you will "bust" or exceed 21, if you take a hit. This is an automatic loss. However, if your total is small, such as a 3 and a 2, then you should take a hit in order to get closer to 21. Knowing when to ask for a hit and when not to is the essence of the game. In making this decision an expert will consider not only the total in his own hand and the card the dealer has showing, but also the cards that have been played in previous hands. This will tell him more accurately what the odds are of certain cards turning up next. Players who are good at remembering which cards have been played are called "card counters", and the most successful ones can actually "beat the odds", and win consistently. (If they do this too obviously, of course, they tend to be banned from the casinos, which are, after all, in business to make money.)

When the player no longer wants to be hit, the dealer then exposes his "hole card" (the one originally dealt face down), and then continues to draw

cards until he reaches a total of 17 or more. At this point the player is informed whether he has won or lost.

Doubling

After he has been dealt the first two cards, the player has the option of doubling his bet and electing to take only one more card. He must decide whether to do this on the basis of the two cards he has showing, and the one card the dealer has showing.

Insurance

If the dealer has an ace showing, the player is given the option of placing an "insurance" side bet equal to one-half the original bet. This side bet pays 2 to 1 if the dealer has blackjack; otherwise, it loses.

Splits

Two cards of equal value dealt to the player are called a "split". A player with a split can choose to split his cards, creating two hands of equal value. Each of these hands will then have a bet equal to the original bet riding on it. Both hands may then win, or one may win and the other lose, or they both may lose. Only one split is possible in a given hand.

Using the Program

When you first run the program, it pauses a few seconds for "initialization" — setting up various arrays. Then it asks you the size of your starting bankroll (in dollars, unless you imagine yourself playing in Monaco, where it would be francs). This can be any amount you like, but, of course, you'll make more of an impression on the crowd if you have a six- or seven-figure bankroll.

You're then asked for your first bet. This should be some reasonable percentage of your bankroll — maybe a tenth. After this the program takes a few seconds to shuffle the deck; then it displays the cards in the first hand. The dealer has one card showing and another face down, indicated by a black square. The player has two cards showing.

The value and suit of each card are indicated. These are self-explanatory, except that "T" is used in place of 10. The screen also displays the size of the player's bankroll and a message asking the player to make a decision — either to double his bet or not, to be hit or not, whether he wants insurance or not, and so forth. The question asked depends on the fall of the cards.

The player responds to these questions by typing "y" for yes or "n" for no, at which point the program takes over and deals additional cards, calculates the new bankroll, or whatever is appropriate.

Warning! This program is addictive and may be hazardous to your health if you make a habit of staying up all night to play it.

The Program Listing

```
10 ' JACK
20 '
30 ' Initialization
40 '
50 CLEAR 400
60 DIM C$(52),H$(2,10),H(2,10),D(2),T(2),PR(2)
70 L$=RIGHT$(TIME$,2)
80 FOR I=1 TO VAL(L$):R=RND(1):NEXT I
90 '
100 ' Define Cards with Graphic Characters
110 '
120 B$="23456789TJQKA":FOR SUIT=0 TO 3:FOR I=1 TO 13:
    A$=A$+MID$(B$,I,1)+CHR$(156+SUIT):NEXT I:NEXT SUIT
130 FOR I=1 TO 52: C$(I)=MID$(A$,2*I-1,2):NEXT I
140 CLS:J=52
150 INPUT "What is your starting bankroll";S
160 INPUT "Bet";B
170 IF J<40 THEN 300
180 '
190 ' Shuffle routine
200 '
210 CLS:J=0
220 PRINT@80,"Shuffling..."
230 FOR I=52 TO 1 STEP -1
240 R=INT(RND(1)*I+1)
250 T$=C$(I):C$(I)=C$(R):C$(R)=T$
260 NEXT I
270 '
280 ' Print Initial Hands
290 '
300 CLS:NS=0:SP=1
310 FOR I=0 TO 2:D(I)=0:T(I)=0:NEXT I
320 PR(0)=40:PR(1)=120:PR(2)=160
330 PRINT @240,"Bet:";B;"   ";"Bankroll:";S
340 PRINT@0,"Dealer:":PRINT@80,"Player:"
350 K=1:GOSUB 1130:PRINT@120,H$(1,1)
360 K=0:GOSUB 1130:PRINT@40,H$(0,1)
370 K=1:GOSUB 1130:PRINT@123,H$(1,2)
380 K=0:GOSUB 1130:PRINT@43,CHR$(239);CHR$(239)
390 '
400 ' Check for BlackJacks and Insurance
410 '
420 IF T(1)<>21 THEN 450
430 S=S+(B/2)
```

```
440 PRINT @160,"Player BlackJack    "
450 IF H(0,1)<>11 AND H(0,1)<>1 THEN 490
460 PRINT @200,"Insurance (Y or N)? "
470 I$=INKEY$:IF I$="" THEN 470
480 IF I$="y" THEN NS=1
490 IF T(0)<>21 THEN S=S-(B/2)*NS:GOTO 540
500 PRINT@43,H$(0,2)
510 PRINT @160,"Dealer BlackJack    "
520 IF T(1)=21 THEN S=S-(B/2)+NS*B:GOTO 1000
530 S=S+NS*B:GOTO 1000
540 IF T(1)=21 THEN 1000
550 '
560 ' Pair splitting
570 '
580 K=1:IF H(1,1)=1 THEN 600
590 IF H(1,1)<>H(1,2) THEN 700
600 PRINT@200,"Split (Y or N)?";STRING$(25," ")
610 I$=INKEY$:IF I$="" THEN 610
620 IF I$<>"y" THEN 700
630 IF H(1,1)=1 THEN H(1,1)=11
640 H(2,1)=H(1,2):T(1)=H(1,1):T(2)=H(2,1)
650 H$(2,1)=H$(1,2):PRINT@123,"  ":PRINT@160,H$(2,1)
660 PR(1)=120:SP=2:GOTO 840
670 '
680 ' Doubling Down
690 '
700 PRINT@200,"Double (Y or N)?";STRING$(24," ")
710 I$=INKEY$:IF I$="" THEN 710
720 IF I$<>"y" THEN 790
730 B=2*B:PRINT@244,B
740 K=1:GOSUB 1130:PRINT@PR(K),H$(1,3)
750 IF BUST=0 THEN 920 ELSE T(K)=-1:GOTO 1000
760 '
770 ' Hitting
780 '
790 PRINT@200,"Hit (Y or N)?";STRING$(27," ")
800 I$=INKEY$:IF I$="" THEN 800
810 IF I$="y" THEN 840
820 IF SP=2 THEN K=2:SP=3:GOTO 840
830 GOTO 920
840 GOSUB 1130:PRINT @PR(K),H$(K,D(K))
850 IF BUST=0 THEN 790
860 T(K)=-1
870 IF SP=2 THEN K=2:SP=3:GOTO 840
880 GOTO 1000
890 '
900 ' Play Dealer's Hand
910 '
920 PRINT@ 43,H$(0,2)
```

```
930 IF T(0)>=17 THEN 1000
940 K=0:GOSUB 1130:PRINT@PR(0),H$(0,D(0))
950 IF T(0)<17 THEN 940
960 IF BUST=1 THEN T(0)=0
970 '
980 ' Compare Player's and Dealer's Hands
990 '
1000 FOR K=1 TO 2
1010 IF T(K)=0 THEN 1050
1020 IF T(K)>T(0) THEN R$="Player wins":S=S+B:GOTO 1040
1030 IF T(K)=T(0) THEN R$="Tie" ELSE R$="Player loses":S=S-B
1040 PRINT @ (PR(K)+5),R$
1050 NEXT K
1060 PRINT @200,STRING$(40," ")
1070 PRINT @43,H$(0,2)
1080 PRINT @240,"              Bankroll:";S
1090 GOTO 160
1100 '
1110 ' Subroutine to Deal a Card, Compute Value and Total Hand
1120 '
1130 D(K)=D(K)+1:BUST=0
1140 J=J+1:K$=C$(J):Z$=LEFT$(K$,1)
1150 IF Z$="A" THEN V=11:GOTO 1170
1160 IF Z$="T" OR Z$="J" OR Z$="Q" OR Z$="K"
     THEN V=10 ELSE V=VAL(Z$)
1170 H$(K,D(K))=K$:H(K,D(K))=V:T(K)=T(K)+V
1180 IF T(K)<=21 THEN 1260
1190 '
1200 ' Check for "Soft" Hands
1210 '
1220 FOR I=1 TO D(K)
1230 IF H(K,I)=11 THEN H(K,I)=1:T(K)=T(K)-10:GOTO 1260
1240 NEXT I
1250 BUST=1
1260 PR(K)=PR(K)+3
1270 RETURN
```

How the Program Works

The characters that make up the deck of cards are defined in line 120. This is done by setting up an array of values, from 2 through 9 plus T, J, Q, K, and A. These are then combined with four numbers 0 to 3, representing the four suits: spades, clubs, diamonds, and hearts. Thus A0 represents the ace of spades, 32 is the three of hearts, and so on. The resulting 52 string values are stored in string array C$, in line 130.

Shuffling the cards represents something of a challenge. In the initialization part of the program the random number generator is first exercized by calling up a number of random numbers; this number depends on the time as obtained from the built-in TIME$ function. The idea here is to make sure that the next random number generated is not always the same when the program is first loaded. Then, in line 240, random number R, from 0 to 51, is generated. This number is used to exchange two numbers, or cards, in the array. For instance, if the number is 15, the program exchanges the A(15) and the A(0), in line 250. Then another random number is generated and the contents of that element, A(R), are exchanged with the A(1). This process continues until the deck is completely shuffled. Array C$ now contains the cards in the order they will be dealt.

Whenever a card is dealt, the subroutine at lines 1110 to 1180 is called. This subroutine computes the value of the card: "A" = 11, "T" = 10, and so on. This value is then added to the value of the cards already dealt to the player, or to the dealer (line 1170). If the resulting value T exceeds 21 (line 1180) then the player (or dealer) has busted. There is one exception: if a player has an ace in his hand, the program recomputes the value of the hand using 1 for the ace instead of 11. The result will be less than 21, so the player will still be "alive," and play continues.

The program starts off by dealing the initial hands, repeatedly calling the "deal a card" subroutine from lines 350 to 380. Then it works its way through a series of routines that check for various situations, which may result from the initial deal. First, in lines 400 to 540, the program checks for blackjacks and insurance, using the value T for the player's hand. If the dealer or the player has blackjack, the result is printed out.

In lines 560 to 660 the program checks for splits. If it finds two cards of equal value, it asks the user if he wants to "split". If so, it sets up two hands for the player, instead of one.

Lines 700 to 750 ask the player if he wants to double, while lines 790 to 880 are responsible for dealing another card when the player asks for a hit.

The final routine executed by many of the paths through in this program is the one from lines 980 to 1090, that compares the values in the player's and the dealer's hands. This section of the program is invoked whenever a hand has been played, and neither the dealer nor the player has won by getting blackjack.

STOMP

Star Stomp

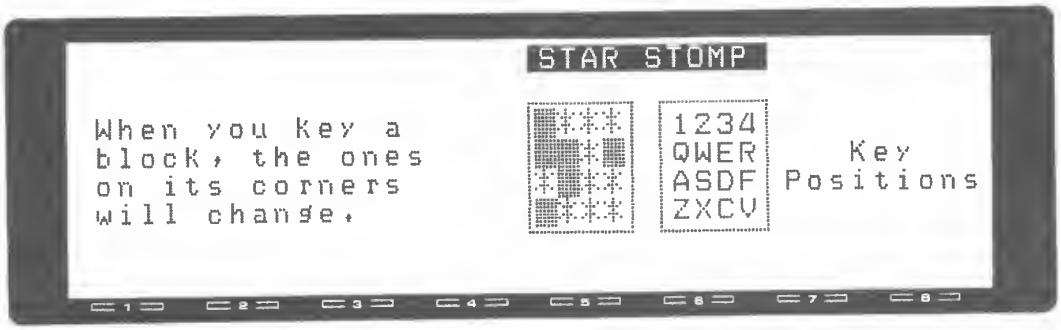
New and Challenging

Looking for a fascinating mental challenge on your Model 100? Stomp those stars! Here's a game that will test your puzzle-solving skills in an entirely new way. The object is to "stomp" out all the stars. The trouble is, when you stomp out one star, more of them pop into view.

When you run the program, two four-by-four grids will be displayed. One contains stars and blocks in random positions; the other is a "key" position diagram that shows which keyboard keys correspond to the positions on the star grid. The object is to "stomp" out all the stars and replace them with blocks. When you enter a valid key, the corresponding star-position will be filled (usually) with a star. Occasionally, however, it will be filled with a block. (This keeps you on your toes.) At the same time, the blocks and/or stars on the *four corners* of your selected position will "flip"; that is, stars will be replaced with blocks and blocks with stars. Sound easy? It's harder than you think!

If you want a change of pace, try filling the grid with stars instead. This is easier than star-stomping and provides a confidence builder, if you want to ease into the game gradually.

To run the program again, press any of the first five function keys (F1), (F2), (F3), (F4), (F5)). You'll get a new board. Every run is a new challenge



because the starting display is randomly generated. But you can "stomp" the stars no matter what board comes up!

For an added challenge, see how few key presses it takes to complete the game, or time yourself. By comparing numbers of moves or times, you can make Star Stomp into a two-person, competitive game which tests not only your skill, but your relationship with lady luck!

Program Listing

```
10 '
20 ' STOMP
30 '
40 'Initial Screen and Definition of String Variables
50 '
60 CLS:DIM V(16),C$(16)
70 CALL 17001:PRINT@17," STAR STOMP ":CALL17006
80 B$=CHR$(239):BL$=CHR$(32):S$=CHR$(42)
90 GOSUB 750
100 '
110 'Read Data into Variables
120 '
130 FOR X=1 TO 16:READ V(X):NEXT
140 FOR X=1 TO 16:READ C$(X):NEXT
150 '
160 'Display Game and Key Location Boards
170 '
180 FOR X=1 TO 16:PRINT@V(X),S$:PRINT@V(X)+6,C$(X):NEXT
190 R=INT(RND(1)*10+.5)
200 '
210 'Random Block Display Loop
220 '
230 FOR R1=1 TO R
240 R2=INT(RND(1)*16+.5):IF R2=0 THEN R2=1
250 PRINT@V(R2),B$:NEXT
260 '
270 'Display Frames and Prompts
280 '
290 LINE (106,14)-(133,49),1,B
300 LINE (141,14)-(169,49),1,B
310 PRINT@80,"When you key a";
320 PRINT@120,"block, the ones";
330 PRINT@160,"on its corners";
340 PRINT@200,"will change.";
350 PRINT@153,"Key";:PRINT@190,"Positions";
360 '
370 'Grid Location Input
380 '
390 I$=INKEY$:IF I$="" THEN 390
400 V%=(INSTR("ZzAaQq11XxSsWw22CcDdEe33VvFfRr44",I$)+1)/2
410 IF V%<1 THEN 320
420 P=65024!+V(V%)
430 '
440 '85/15% Chance of Star/Block Display
450 '
```

```
460 RN=RND(1)
470 IF RN<.85 THEN PRINT @ V(V%),B$; ELSE PRINT @ V(V%),BL$;
480 '
490 'Determines Positions of Corners
500 '
510 P1=P-41:V1=V(V%)-41:GOSUB 600
520 P1=P-39:V1=V(V%)-39:GOSUB 600
530 P1=P+39:V1=V(V%)+39:GOSUB 600
540 P1=P+41:V1=V(V%)+41:GOSUB 600
550 '
560 'Determines Grid Selected and Valid Location Check
570 '
580 P1=P:V1=V(V%):GOSUB 600
590 GOTO 390
600 IF P1>65121! AND P1<65126! THEN 680
610 IF P1>65161! AND P1<65166! THEN 680
620 IF P1>65201! AND P1<65206! THEN 680
630 IF P1>65241! AND P1<65246! THEN 680
640 RETURN
650 '
660 'Display Reverse Corners (Blank/Block)
670 '
680 L=PEEK(P1)
690 IF L=239 THEN PRINT@ V1,S$;:RETURN
700 IF L<>239 THEN PRINT@ V1,B$;
710 RETURN
720 '
730 'Random Game Board Generation
740 '
750 FOR I=1 TO VAL(RIGHT$(TIME$,2))
760 RN=RND(1)
770 NEXT: RETURN
780 '
790 'Screen Position and Key Character Data
800 '
810 DATA 218,178,138,98,219,179,139,99,220,180,140,100,221,181,141,101
820 DATA Z,A,Q,1,X,S,W,2,C,D,E,3,V,F,R,4
```

How the Program Works

Line 60 clears the screen and establishes the string variables V and C$. The game title is displayed, using assembly calls to reverse the background and foreground colors on the screen in line 70. ASCII values for B$ (block), BL$ (blank), and S$ (star) are given in line 80. The subroutine in lines 750 through 770 generates a random game board for each run.

The program uses the two arrays, V and C$, to hold sixteen positions and the sixteen corresponding keys for the two four-by-four grids. The DATA statements in lines 810 and 820 hold the data for the two strings. Lines 180 through 350 set up and display the game and key location boards.

The central routine in lines 390 through 420 waits for input, and then begins the grid location determination for the entered value. The entry is checked to see that it is valid within the established grid in lines 600 through 630. If it is, then the program PEEKS to see what the values for the corners were previously and then switches them (lines 680 through 710). The square selected is randomly displayed with an 85% chance that it will be a star, and a 15% chance that it will be a block.

5

Fun with Sound and Music

MORSE

Morse and Remorse

Morse Code Generator

Are you looking for a job with a top-secret spy agency? Would you like to be able to understand those mysterious "dot-dash" messages you hear on your short wave radio? Have you always wanted to be a ham radio operator, but couldn't learn the Morse code? This program will enable you to send flawless code, any time you want, with no practice at all! You can transmit at speeds that would dazzle an expert, or you can slow down your transmissions and use the program to teach yourself the code at your own pace. This program generates all the letters, numbers, and standard punctuation, so it can be used in any situation where Morse is called for.

In addition, we tell you how to take the output of this program and send it through the cassette port, so that it can be fed directly to your ham rig or clandestine transmitter.

When you first start the program, it asks you how fast you want to transmit in words per minute: SPEED (WPM). You respond by entering the appropriate number. Five words per minute is slow; it's called "novice class" in amateur radio. Twenty WPM is "extra class", the highest level. Professional operators can transmit even faster.

Next, the program says "MESSAGE". Here you type in the text of the message you want to send. This can be anything from a single character, up

to 255 characters. As soon as you press (ENTER), the program will start to transmit the message. You'll hear it load and clear from the Model 100's speaker.

Transmitting Through Your Cassette Port

It's possible to hook up your Model 100 to a radio transmitter and use this program to transmit your Morse code message anywhere in the world. This requires building a small relay circuit, so if you aren't handy with pliers and a soldering iron, you can skip this section.

To get the Morse signal out of the Model 100, we modify the program to send the signal through the cassette port instead of the speaker. However, since the relay inside the Model 100 that sends this cassette port signal is very small, it's important not to use it to drive your transmitter directly. Instead, use the cassette output to drive another relay, using the circuit shown in Figure 5-1. The signal is the one used to turn the cassette motor on and off. It is found in the REM (Remote) plug in the cassette cable.

Construct the relay box using any suitable enclosure that's large enough to hold the components. You might want to use a battery holder with four AA batteries for a power supply. This will give you approximately 6 volts to drive the relay. The relay specified in the parts list in the figure draws only 12 mA of current when activated, so these batteries should last quite a while. The zener diode is connected across the leads to the relay coil to reduce any arcing that might damage the contacts of the relay inside your Model 100.

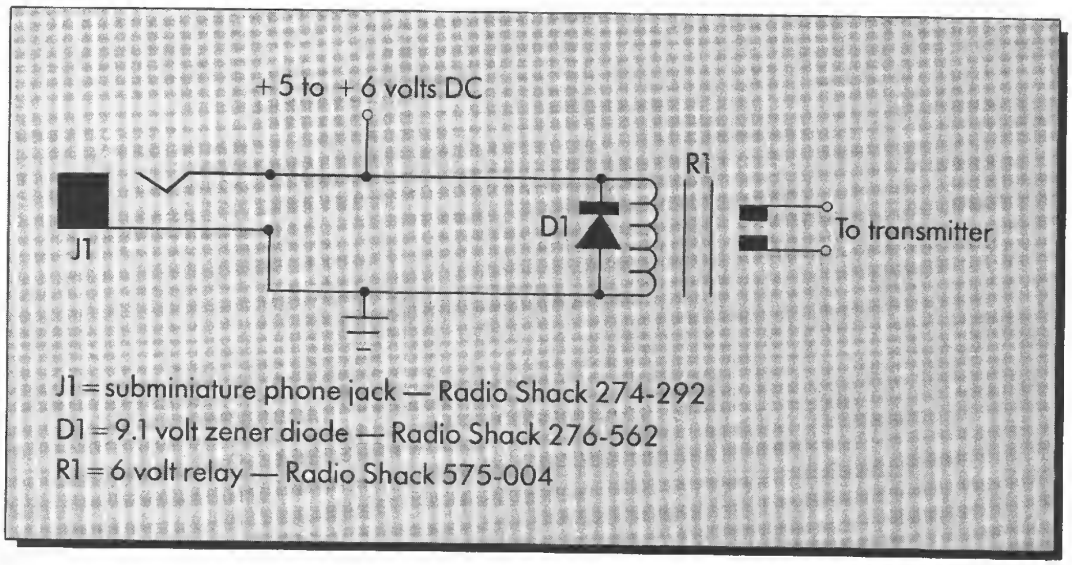


Figure 5-1. Morse code relay circuit with parts list

To use the relay box, plug the regular cassette cable into the cassette port on the back of your Model 100; then plug the smaller of the two gray plugs on the other end of the cable into the jack on the relay box. The connection to the transmitter has not been specified; this will depend on your particular transmitter. Remember that your transmitter keying circuit should stay within the 1 amp range of the relay box.

To route the output of the program to the cassette port instead of the speaker, the program needs to be modified slightly. Substitute the following lines for 180 and 190:

```
180 IF C$="S" THEN MOTOR ON:GOSUB 270:MOTOR OFF:GOSUB 270

190 IF C$="L" THEN MOTOR ON:GOSUB 280:MOTOR OFF:GOSUB 270
```

Program Listing

```
10 '
20 ' MORSE
30 '
40 CLEAR 300:CLS:DIM A$(90)
50 READ A$(44):FOR X=46 TO 57:READ A$(X):NEXT
60 READ A$(63):FOR X=65 TO 90:READ A$(X):NEXT
70 INPUT"SPEED (WPM) ";S
80 S=1/S:S=S*500
90 LINE INPUT"MESSAGE: ";M$:L=LEN(M$)
100 FOR X=1 TO L
110 '
120 'Translate Message to Morse
130 '
140 B$=MID$(M$,X,1):C=ASC(B$):IF C>90 THEN C=C-32
150 IF C=32 THEN GOSUB 290:X=X+1:GOTO 140
160 FOR Y=1 TO LEN(A$(C))
170 C$=MID$(A$(C),Y,1)
180 IF C$="S"THEN SOUND 2000,2:GOSUB 270
190 IF C$="L"THEN SOUND 2000,6:GOSUB 270
200 'IF X=L THEN GOSUB 290
210 NEXT Y:GOSUB 280:NEXT X: GOTO 90
220 '
230 'character codes and delay routines
240 '
250 DATA LLSSLL,SLSLSL,LSSLS,LLLLL,SLLLL,SSLLL,SSSLL,SSSSL,SSSSS,
    LSSSS,LLSSS,LLLSS,LLLLS,SSLLSS,SL,LSSS,LSLS,LSS,S,SSLS,LLS
260 DATA SSSS,SS,SLLL,LSL,SLSS,LL,LS,LLL,SLLS,LLSL,SLS,SSS,L,SSL,
    SSSL,SLL,LSSL,LSLL,LLSS
270 FOR Z=1 TO S:NEXT:RETURN
280 FOR Z=1 TO S*3:NEXT:RETURN
290 FOR Z=1 TO 7*S:NEXT:RETURN
```

How the Program Works

The heart of this program is the way the Morse code characters are stored in DATA statements in lines 250 and 260. Each character is stored as a string constant, with "L" representing a long tone, or dash, and "S" representing a short tone, or dot. Thus the first character, "LLSSLL", representing a comma (,), sounds like dah dah di di dah dah. Lines 50 and 60 in the program read these string constants from the DATA statements and store them in array A$. The characters are stored in such a way that the subscript for each character is the same as the ASCII code of the character. Thus the comma, with an ASCII code of 44, is stored at A$(44). The relationship of the Morse characters and their ASCII codes is shown in Table 5-1.

Lines 70 to 90 get the speed and the message from the user. The message is stored in variable M$. Line 40 then uses a MID$ statement to remove one character at a time from M$.

The three subroutines in lines 270, 280, and 290 generate three different time intervals. These intervals are all related to the value of S, the speed typed in by the user. If a character in the message M$ is a space (as determined in line 150), then the longest of the time intervals, line 290, is used. Otherwise, the program looks up the character in the array A$ to get the Morse string variable (like SSLLSS). This is assigned to variable C$. Lines 160 to 210 then take apart the C$ variable to send the individual dots and dashes, using the SOUND statement.

Symbol	Array Location	Morse Variables
comma (,)	A$(44)	LLSSLL
period (.)	A$(46)	SLSLSL
slash (/)	A$(47)	LSSLS
numbers 0 to 9	A$(48) to (57)	LLLLL, etc.
question mark (?)	A$(63)	SSLLSS
Letters A to Z	A$(65) to (90)	SL, etc.

Table 5-1. Relationship between Morse characters and their ASCII codes

PIANO

Portable Piano

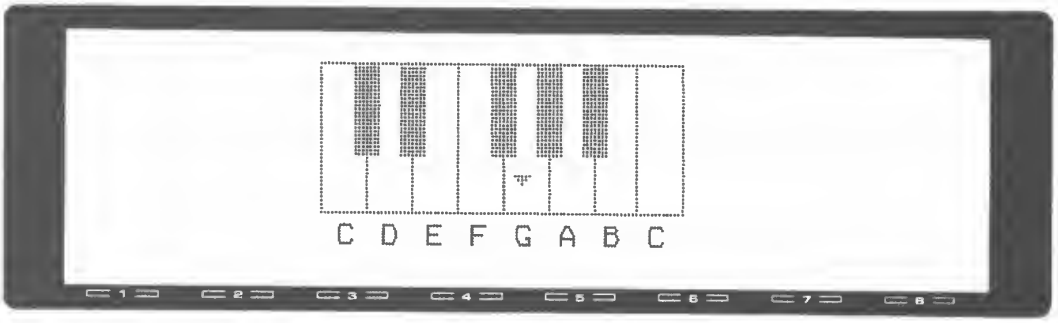
Keyboard in Your Briefcase

Imagine yourself, wearing black tie and tails, seated at the keyboard. You are the featured performer of the evening, all alone on the stage. You are introduced, the crowd falls silent, and a single white spotlight focuses on your hands as they hover over the keys. And what famous and sophisticated instrument have you chosen for your debut at Carnegie Hall? Why, the Model 100 and this Piano program — what else?

Piano converts your Model 100 into a miniature piano keyboard. It may not have the full rich tone of a Steinway, but, on the other hand, you can't carry a Steinway in your briefcase. In addition to playing notes, when you press the appropriate keyboard keys, it also displays a picture of a piano keyboard on the screen and shows which note you're playing by drawing a small triangle on the note.

This program can be used for fun, or to teach music fundamentals, since you can see at a glance the name of the note you're playing.

The white notes are represented by the keys A,S,D,F,G,H,J,K. These are the middle row of keys. The black notes are represented by the keys W,E,T,Y,U, located on the upper row of keys. The locations of these letters correspond, as closely as possible, to the arrangement of notes on a real piano keyboard.



The octave can also be changed so that it is possible to play any note within the range of the computer's sound system. This is done with the cursor control keys — the [↑] key changes to a higher octave, and the [↓] key changes to a lower octave. The total possible range is about seven octaves, almost as great as a real piano. By using these octave keys, it is possible to play any melody, even one that extends over several octaves. Be sure, however, that the [CAPS LOCK] key is not depressed. This program will only recognize lower case letters.

The Program Listing

```
10 '
20 '  PIANO
30 '
40 DIM A(25),B(13),P(25),K(13)
50 DATA 9394,8866,8368,7900,7456,7032,6642,6269,
   5918,5586,5272,4976,4697
60 DATA 1,23,19,5,4,6,20,7,25,8,21,10,11
70 DATA 171,92,173,94,175,177,98,179,100,181,102,183,185
80 '
90 ' Store Notes and Print Positions of Keys
100 '
110 FOR I=1 TO 13:READ B(I):NEXT I
120 FOR I=1 TO 13:READ K(I):J=B(I):A(K(I))=J:NEXT I
130 FOR I=1 TO 13:READ Q:P(K(I))=Q:NEXT I
140 '
150 ' Draw Piano Keyboard
160 '
170 CLS: LINE (63,4)-(159,44),1,B
180 FOR L=75 TO 147 STEP 12
190 LINE (L,5)-(L,43):NEXT L
200 FOR P=72 TO 132 STEP 12
210 IF P=96 THEN 230
220 LINE (P,4)-(P+6,28),1,BF
230 NEXT P
240 PRINT@251,"C D E F G A B C"
250 '
260 ' Play Notes and Display on Screen
270 '
280 GBOX$=CHR$(239):GTRI$=CHR$(167)
290 M=1:PP=1
300 N$=INKEY$:IF N$="" THEN 300
310 C=ASC(N$)-96
320 IF C=-66 THEN M=M/2
330 IF C=-65 THEN M=2*M
340 IF C>25 OR C<0 THEN 300
350 IF P(C)=0 THEN 300
360 FR=A(C)*M:IF FR>16383 THEN 300
370 IF PP/2-INT(PP/2)=0 THEN PRINT @PP,GBOX$:GOTO 390
380 PRINT @PP," "
390 PRINT@P(C),GTRI$
400 SOUND FR,10
410 PP=P(C):GOTO 300
```

How the Program Works

This program uses the SOUND statement to generate notes. The numerical values to use with this statement to produce a particular note are listed in the *TRS-80® Model 100 Portable Computer* manual, in the BASIC section under the SOUND statement. These values are called "pitch values". The central problem to be solved in the Piano program is to figure out what pitch value to use when a particular key is pressed.

This program would have been simpler to write if sharps and flats had not been used. Then the number keys — on the top row of the keyboard — could have represented the notes, and there would have been a simple relationship between the keyboard keys and the pitch values since the ASCII values of the number keys run in order from 48 to 57. However, as it is, there is no simple relationship between the ASCII value of the key pressed for a particular note and the note itself.

This problem is dealt with in the following way. An array B is first set up (lines 50 and 110) which contains the pitch values. These values are then transferred to a different array, A, which consists of one value for each letter of the alphabet we're using, from "a" (element number 1) to "y" (element number 25). This is done in lines 60 and 120. Later, in line 320, when a key is pressed, the proper element in array A is found by subtracting 96 from the ASCII code for the key.

A small triangle, superimposed on the keyboard, is used to show which key is being pressed. Lines 70 and 130 assign the print positions for this triangle. These are also stored in array A.

Lines 170 to 240 draw the piano keyboard on the screen, using the LINE statement. Line 280 assigns variable names to several graphics symbols. In lines 320 and 330 the program checks to see if the cursor control keys are being pressed. If so, the frequency of the note is either halved or doubled, to lower or raise the octave. Finally, line 400 plays the note.

NOTE

Noterony

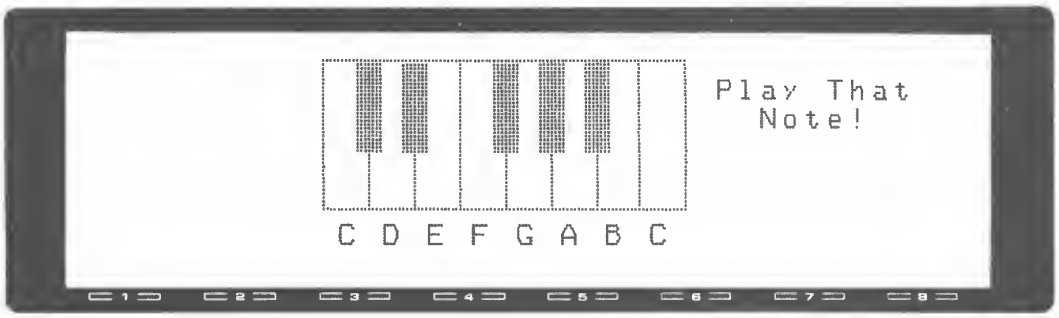
Note Guessing Program

*H*ave you ever wanted to play first violin for the New York Philharmonic? Do you envy Arthur Rubenstein and Isaac Stern? Well, it's not too late to start your musical education!

This program may not be the equivalent of a complete course at Julliard, but it will allow you to practice and improve your musical skills by guessing notes the program plays for you. It acts as your own personal music teacher, playing a note and then giving you an opportunity to guess the note by playing it on your simulated Model 100 piano keyboard. It then informs you whether you're right or wrong, and, if you're wrong, it gives you an opportunity to guess again. You can use it to develop your musical ear and increase your understanding of scales and the relationship between notes.

This program can also be used to play tunes in the same way the Piano program can.

To activate the note guessing part of this program, you simply type the number "1". The program will then respond by playing a note, while displaying the message "Play that Note!" (See the description of the Piano program to find out which keyboard keys apply to which notes.) You play the note, and if you guess wrong, the program flashes the message "Wrong!



Try Again!" If you're right, you'll see the message "Right! Very Good!" You can start over by typing the number "1".

A substantial part of this program is the same as the Piano program. Thus, if you have already typed in the Piano program, you don't need to type in lines 10 to 400. Simply add lines 410 to 590 from the listing below. In addition, line 300 must be changed as shown in the listing.

Program Listing

```
10 '
20 '  NOTE
30 '
40 DIM A(25),B(13),P(25),K(13)
50 DATA 9394,8866,8368,7900,7456,7032,6642,6269,5918,5586,5272,4976,4697
60 DATA 1,23,19,5,4,6,20,7,25,8,21,10,11
70 DATA 171,92,173,94,175,177,98,179,100,181,102,183,185
80 '
90 ' Store Notes and Print Positions of Keys
100 '
110 FOR I=1 TO 13:READ B(I):NEXT I
120 FOR I=1 TO 13:READ K(I):J=B(I):A(K(I))=J:NEXT I
130 FOR I=1 TO 13:READ Q:P(K(I))=Q:NEXT I
140 '
150 ' Draw Piano Keyboard
160 '
170 CLS: LINE (63,4)-(159,44),1,B
180 FOR L=75 TO 147 STEP 12
190 LINE (L,5)-(L,43):NEXT L
200 FOR P=72 TO 132 STEP 12
210 IF P=96 THEN 230
220 LINE (P,4)-(P+6,28),1,BF
230 NEXT P
240 PRINT@251,"C D E F G A B C"
250 '
260 ' Play Notes and Display on Screen
270 '
280 M=1:PP=1
290 N$=INKEY$:IF N$="" THEN 290
300 IF N$="1" THEN G=1:GOSUB520:GOTO290
310 C=ASC(N$)-96
320 IF C=-66 THEN M=M/2
330 IF C=-65 THEN M=2*M:
340 IF C>25 OR C<0 THEN 290
350 IF P(C)=0 THEN 290
360 FR=A(C)*M:IF FR>16383 THEN 290
370 IF PP/2-INT(PP/2)=0 THEN PRINT @PP,CHR$(239):GOTO 390
380 PRINT @PP," "
390 PRINT@P(C),CHR$(167)
400 SOUND FR,10
410 PP=P(C):IF G=0 THEN 290
420 '
430 ' Note-Guessing Program
440 '
450 '
```

```
460 IF GR=FR THEN PRINT@68,"Right!   ":
    PRINT@110,"Very Good!":G=0:GOTO 290
470 PRINT@68,"Wrong!   "
480 PRINT@110,"Try Again!"
490 GOTO 290
500 '
510 ' Generate Random Note
520 '
530 PRINT@68,"Play That"
540 PRINT@110,"Note!     "
550 IF PP/2-INT(PP/2)=0 THEN PRINT @PP,CHR$(239):GOTO 570
560 PRINT @PP," "
570 R=INT(RND(1)*13)+1
580 GR=A(K(R)):SOUND GR,10
590 RETURN
```

How the Program Works

The first part of this program is the same as the Piano program shown earlier. Please refer to that description for lines up to 400. In line 300 the program checks to see if you have typed a number "1". If so, a flag G is set to 1 so that other parts of the program will be aware that the program is in "note guessing" mode. Then the subroutine at line 510 is called to generate a random note. This subroutine prints the "Play that Note!" message, generates a random number between 1 and 12, and plays the corresponding note. The program then returns to line 290 to continue scanning the keyboard for a note.

After the user plays a note, line 410 will check to see whether the program is in normal mode or note-guessing mode. If in normal mode, the flag G will be 0, and the program will go back to scan for another note. However, if G = 1, then the program is in note-guessing mode, and the instructions from 430 to 490 will be executed. These check to see if the note played by the computer is the same as the one played by the user (line 460). If so, the "That's Right!" message is displayed; if not, the user will see "Wrong! Try again!" on the screen.

6

Fun with Pictures

DRAW

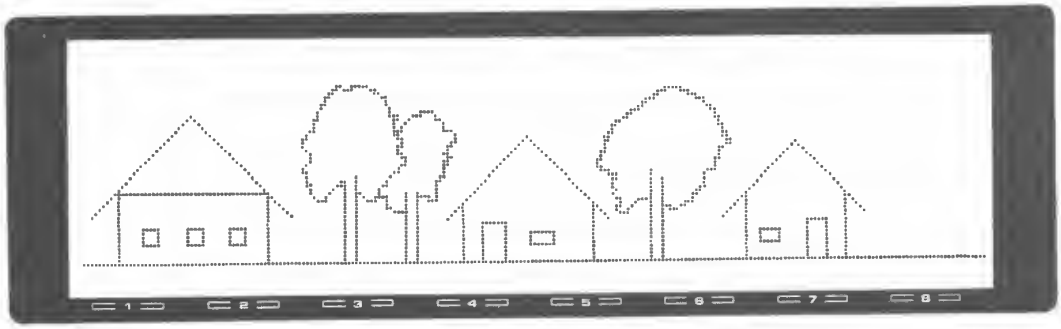
Micro-Rembrandt

Drawing Game

Everyone likes to doodle. Some people even get quite good at it and go on to be famous artists, as Rembrandt himself did. However, doodling with pencil and paper has become old-fashioned, passé, and socially unacceptable. Is there a way you can doodle in public — at board meetings or in your first-class seat on the Concorde — and still look suave, sophisticated, and knowledgeable? Of course, with Micro-Rembrandt! This program lets you create designs and drawings with a surprising degree of detail and nuance right on your Model 100! No one will suspect you're doodling on your computer; they'll think you're writing a program or analyzing complex financial data.

But don't get the idea that this program is completely frivolous. It can serve as a practice program for its more sophisticated cousin, Mega-Rembrandt, which permits you to save the drawings you create and incorporate them into other programs. (The description of Mega-Rembrandt follows this program.)

Because Micro-Rembrandt is a very short, elegant program, it's easy to key in. Thus, it can be used at almost any time, even if it is not already stored in your computer's memory. In addition, it's an improvement over



pencil and paper because there's no problem in drawing absolutely straight vertical, horizontal, and diagonal lines, even if you forgot your ruler.

How is it possible to draw pictures on your computer? After all, you usually see the screen used for numbers and letters, not pictures. But the screen of the Model 100 can be used in two different modes. The first is the "character mode", which displays letters, numbers, and other characters, and is the one used in most programs. In character mode the screen is divided into 8 rows of 40 characters each, a total of 320 possible positions. The second display mode is the "graphics mode", in which the screen is divided into 64 rows of 240 dots, for a total of 15,360 dot positions on the total screen. In computerese these dots are called "pixels". Micro-Rembrandt lets you turn on or off any of these 15,360 pixels, to create surprisingly detailed pictures.

When the program is first loaded, a small, blinking pixel will appear in the center of your screen. This is the "graphics cursor". To draw a picture, you move this cursor around using the keyboard keys. Nine keys are needed — eight to go in different directions and one to draw the dot at its current location. These nine keys are clustered together on the left side of the keyboard, as shown in Figure 6-1.

To move the cursor, you simply hold down the key for the direction you want to go. The cursor will move and draw a line behind it. If you want to move the cursor without drawing a line, or to erase something already drawn, press the "n" key. The cursor will then be in "erase mode" until you toggle it back to "draw mode" by pressing "n" again. If you're tired of your drawing and want to start a new one, press the number "0". This will clear the entire screen (be careful!) and start the program over.

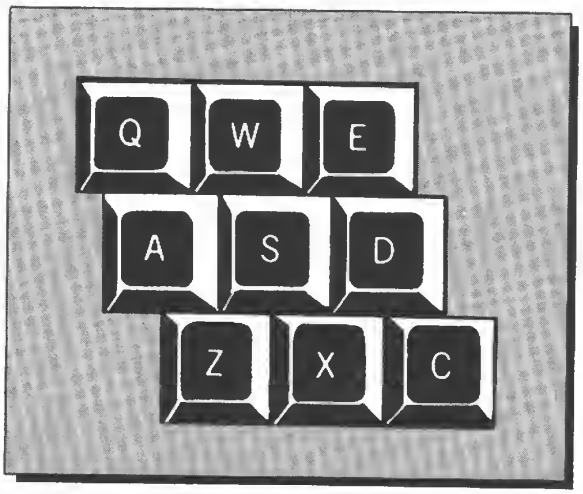


Figure 6-1. Key cluster for graphics cursor

The Program Listing

```
10 '
20 ' DRAW
30 '
40 CLS:I=120:J=32:F=0
50 FOR X=1 TO 50:PSET(I,J):NEXT X
60 FOR X=1 TO 50:PRESET(I,J):NEXT X
70 A$=INKEY$:IF A$="" THEN 50
80 IF A$="0" THEN 40
90 IF A$="n" THEN F=1-F:GOTO 180
100 IF A$="e" OR A$="d" OR A$="c"THEN I=I+1:IF I>239 THEN I=239
110 IF A$="q" OR A$="a" OR A$="z" THEN I=I-1:IF I<0 THEN I=0
120 IF A$="q" OR A$="w" OR A$="e" THEN J=J-1:IF J<0 THEN J=0
130 IF A$="z" OR A$="x" OR A$="c" THEN J=J+1:IF J>63 THEN J=63
140 PSET (I,J)
150 IF F=0 THEN 180
160 FOR X=1 TO 10:NEXT X
170 PRESET (I,J)
180 A$=INKEY$:IF A$="" THEN 180 ELSE GOTO 80
```

How the Program Works

This program consists essentially of a simple loop, occupying the lines from 80 to 180. The keyboard is scanned, and when a key is pressed, it's assigned to the variable A$. The four statements in lines 100 to 130 then figure out which direction to move the cursor.

When the program is first started, lines 50 to 70 position the graphics cursor in the center of the screen and flash it on and off so it's easier to recognize. Line 70 looks for a key to be pressed, and when it finds one, control passes to the main part of the program in line 100.

Lines 100 to 130 are the heart of the program. Each line is concerned with a particular direction. Line 100 looks for the keys "e", "d", and "c". If any of these keys have been pressed, the cursor should be moved to the right. Similarly, line 110 moves the cursor to the left if any of the keys "q", "a", or "z" have been pressed; line 120 moves it upward on "q", "w", or "e"; and line 130 moves it downward on "z", "x", or "c". Since the cursor can move diagonally, two lines in the listing can be active at the same time. Thus if the "q" key has been pressed, both line 110 and line 120 will be active, and the cursor will move both up and to the left.

The variables I and J are used to hold the current position of the cursor. It is these variables that are changed in lines 100 to 130 to move the cursor. Variable I represents the x-coordinate; it is incremented to move the cursor right, and decremented to move it left. Variable J, the y-coordinate, is incremented to move the cursor down, and decremented to move it up. The PSET instruction in line 140 is used to draw the dot on the screen at the new cursor position.

When the "n" key is pressed, a flag F is set to 1 — if it was set to 0 before — to indicate that dots are to be erased instead of drawn. If F was already set to 1, it is set back to zero. This is handled in line 90. When F = 1, line 170 becomes operative, and, each time through the loop, erases the dot which has just been drawn in line 140.

SAVE

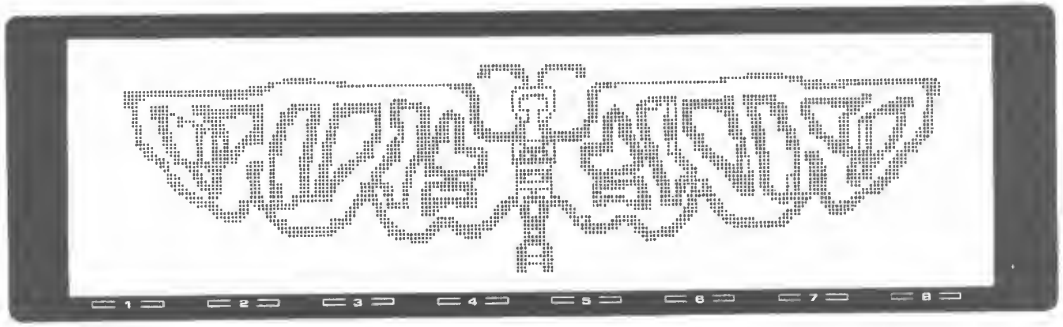
Mega-Rembrandt

Saves Drawings to Memory

Mega-Rembrandt is the professional, serious version of the Micro-Rembrandt program. To learn how to draw pictures with Mega-Rembrandt, you should read the description of Micro-Rembrandt. As far as drawing the picture goes, the two programs function identically. The increased usefulness of Maxi-Rembrandt stems from the fact that the picture which you create can be *saved* in the Model 100's memory, then reloaded onto the screen any time you wish. You can even write the file containing the picture onto a cassette tape, thus creating a more permanent version.

Pictures saved in memory can be called back onto the screen at any time. You can either do this yourself, with this program, or a picture can be called from another program written in BASIC. Many different kinds of programs can profit from the use of illustrations. These might be game programs requiring a sophisticated visual scene, or pictures of some apparatus (a machine tool, for example) for which you are writing an interactive instruction program.

When Maxi-Rembrandt is first started, it asks if you want to "copy a drawing from file". This means, do you want to use a picture which you created before and stored in memory? If so, you answer "y", if not, "n". The first time you use this program, of course, there will be no picture stored,



so you will answer "n". Then you create the picture using the keyboard keys, as described in the Micro-Rembrandt program.

To save the picture, you exit from the drawing part of the program by typing the number 0. The program then asks if you are interested in saving the picture. To save it, you type "y". The program will then ask you to wait while it writes the picture into the file PICT.DO. This takes about ten seconds. Then the program starts over, asking again if you want to read a drawing from the file or create one of your own. You can read the same picture back in or create a new one.

Note that in order to save the picture you must have at least 1,024 free bytes of memory; this is the size of the PICT.DO file. Also, remember that you can only save one picture in memory at any one time. If you want to save more pictures, you must write PICT.DO onto a cassette tape. The pictures are document files, and can be written to tape in just the same way as any other document file. (See *Introducing the TRS-80® Model 100* by Diane Burns and Sharyn Venit [New York: Plume/Waite, New American Library, 1984], for a full description of this procedure.)

Program Listing

```
10 '
20 ' SAVE
30 '
40 DEFINT B-Z:DIM K(1023)
50 INPUT "Copy drawing from file (Y or N)";A$
60 CLS:IF A$="y" OR A$="Y" THEN 130
70 PRINT "wait..."
80 FOR N=0 TO 1023:K(N)=0:NEXT N
90 CLS:GOTO 260
100 '
110 ' Load from File
120 '
130 OPEN "Ram:pict.do" FOR INPUT AS 1
140 FOR BYTE=0 TO 1023
150 INPUT #1,K(BYTE)
160 IF K(BYTE)=0 THEN 220
170 J=INT(BYTE/16):N=BYTE MOD 16
180 FOR BIT=0 TO 14
190 I=N*15+14-BIT
200 IF (K(BYTE)AND(2^BIT))<>0 THEN PSET (I,J)
210 NEXT BIT
220 NEXT BYTE: CLOSE 1
230 '
240 ' Manual Drawing
250 '
260 I=120:J=32:F=0
270 FOR X=1 TO 50:PSET(I,J):NEXT X
280 FOR X=1 TO 50:PRESET(I,J):NEXT X
290 A$=INKEY$:IF A$="" THEN 270
300 IF A$="0" THEN 450
310 IF A$="n" THEN F=1-F:GOTO 410
320 IF A$="e" OR A$="d" OR A$="c"THEN I=I+1:IF I>239 THEN I=239
330 IF A$="q" OR A$="a" OR A$="z" THEN I=I-1:IF I<0 THEN I=0
340 IF A$="q" OR A$="w" OR A$="e" THEN J=J-1:IF J<0 THEN J=0
350 IF A$="z" OR A$="x" OR A$="c" THEN J=J+1:IF J>63 THEN J=63
360 PSET (I,J)
370 BYTE=J*16+INT(I/15):BIT=14-I MOD 15
380 IF F=0 THEN K(BYTE)=(K(BYTE)OR(2^BIT)):GOTO 410
390 K(BYTE)=(K(BYTE)AND(NOT(2^BIT)))
```

```
400 PRESET (I,J)
410 A$=INKEY$:IF A$="" THEN 410 ELSE GOTO 300
420 '
430 ' Save Drawing
440 '
450 CLS:INPUT "Save to file (Y or N)";A$
460 IF A$<>"y" AND A$<>"Y" THEN 50
470 PRINT "Wait...":OPEN "ram:pict.do" FOR OUTPUT AS 1
480 FOR BYTE=0 TO 1023
490 PRINT #1,K(BYTE);
500 NEXT BYTE
510 CLOSE 1:GOTO 50
```

How the Program Works

Much of this program is identical to Micro-Rembrandt. In particular, the lines which scan the keyboard and then adjust the variables I and J to reflect the current position of the cursor are just the same (although the line numbers are different). For a description of this part of the program, read "How the Program Works" in Micro-Rembrandt. Here we'll concern ourselves with the parts of the program that load the PICT.DO file from memory onto the screen and save the screen image back to memory.

To save the picture in memory requires storing 240 times 64, or 15,360 pixels (picture elements). The program uses one bit to represent each pixel. This reduces the storage space required in memory for storing the picture. If a BASIC integer were used to store each pixel, two times 15,360 or 30,720 bytes of memory would be needed, since an integer variable requires 2 bytes, or 16 bits. One of these bits is used for a sign bit and is, therefore, not easily accessible to the program. Thus, 15 bits can be squeezed into each integer variable, so that only 1,024 bytes of memory are required to store the 15,360 pixels on the screen. This is reflected in the DIM statement in line 40, where the array K is set up to hold the information that will be written to memory.

Whenever a pixel is filled in on the screen by the drawing part of the program in line 360, a corresponding bit is set to 1 in the array K in lines 360 to 390. The trick in these lines is to figure out which bit in array K corresponds to a particular pixel. The variable I is the x-coordinate of the pixel, and J is the y-coordinate. In any given byte, the y-coordinates are all the same; that is, a byte contains 15 bits with the same y-coordinate, but different x-coordinates.

The bit is represented by the variable BIT, and the byte by the variable BYTE. (Clever names, eh?) Line 370 calculates BIT and BYTE from I and J. If the flag F is set, then the bit must be turned off again, both on the screen and in array K. This is handled in line 380.

At the beginning of the program the user can elect to read the file PICT.DO from memory onto the screen. This is the opposite process from that just described. Line 130 opens the file, and the loop from 140 to 220 is responsible for reading the file one byte at a time and translating each bit into the corresponding I and J coordinates. The latter operation is handled in lines 170 and 190.

LIFE

Conway's Life

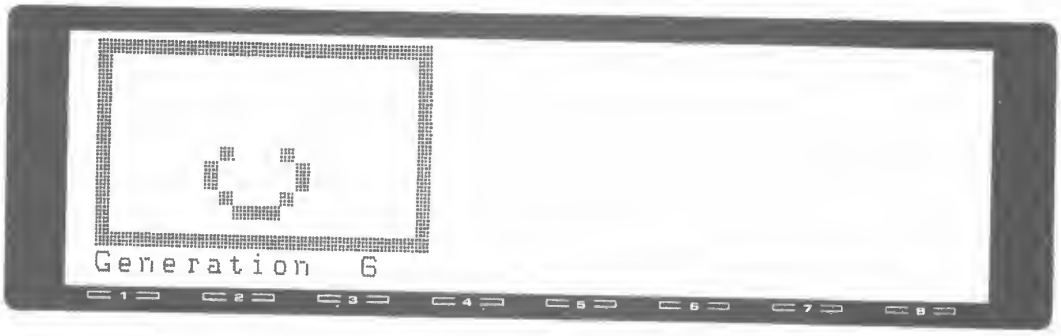
Visual Microbe Colonies

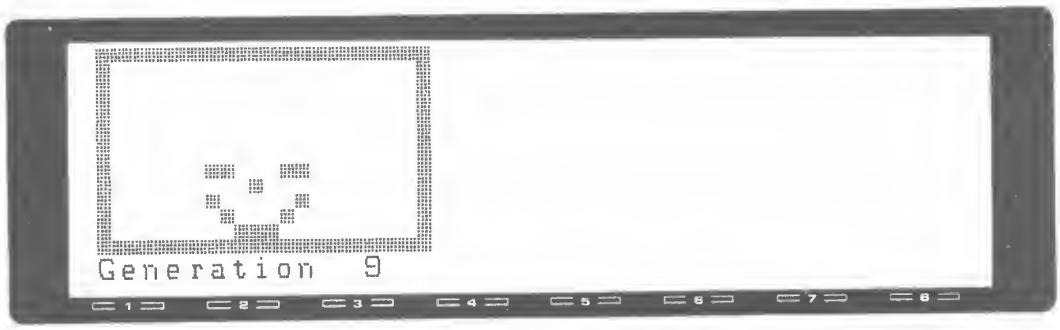
LIFE is a fascinating, non-competitive game which appeals to artists, mathematicians, and anyone who is intellectually curious. It was invented by British mathematician J.H. Conway. The idea of the game is to start with a simple pattern on the screen, and by applying a few simple rules, automatically generate new patterns. The changing patterns, which may become incredibly complex over time, resemble the growth of colonies of living organisms, such as microbes, where individual members are born, live, and die.

In many versions of Conway's Life the starting locations of the individual members must be typed in by hand, in the form of x- and y- coordinates. This version is particularly easy to operate because the starting locations can be filled in using the cursor keys, as if you were drawing a picture.

The Rules of the Game

The game takes place on a board made up of squares like a checkerboard. In this particular version of Conway's Life the player can select the size of the board. A larger board permits more complex patterns, but the game runs more slowly.





Once the size of the board has been selected (as described below), the player inputs the initial arrangement of "cells" or elements by using the cursor keys. This initial arrangement is called the "first generation". The program then takes over and produces the subsequent generations. The changing patterns form a fascinating and hypnotic visual spectacle. It is very difficult to predict in advance how a particular pattern will evolve. Some patterns die out after a few generations. Others fall into a series of patterns which is repeated over and over, or into a single pattern which does not change at all. The most interesting patterns continue to evolve and change, generating one unique picture after another.

The rules the program uses to change each generation into the next are very simple. Each space on the game board can either contain a cell or not. Each cell has eight "neighbors" or adjacent cells — one each above and below, on either side, and in the four diagonal corners. The rules governing the existence of a cell depend on how many neighbors it has:

1. A cell will appear, or be "born", in a vacant square, if it is surrounded by exactly three occupied (or "living") cells.

2. A cell will continue to exist (or "live") if it is surrounded by either two or three living cells.

3. A cell will die if it is surrounded by fewer than two or more than three living cells, as though it dies from loneliness if it has too few companions, or from overcrowding if it has too many.

Operating the Game

When you first run this program, you will be asked for the dimensions of the playing board. A good set of starting dimensions is 12-by-12. You can go as high as 14-by-58, but this runs much more slowly. Simply enter the numbers following the question marks and press ENTER:

```
? 12
?? 12
```

Once you enter the board size, the program will outline the screen, and position a special cursor in the upper left-hand corner. You can move this little cursor around the screen using the cursor control keys: [↑], [↓], [←], [→]. To draw the initial pattern (the "first generation") on the screen, you move the cursor to the square where you want to install a living cell, then press the "1" key (the number "1"). Then you move to the next location and press the "1" again. In this way an entire pattern can be created. If you make a mistake and want to erase a cell, use the "0" key (the number "0").

As soon as your pattern is complete, you can start the generation process by pressing the "L" key (for "LIFE") either upper or lower case.

What Patterns?

What patterns should you start off with? A lot of the fun of this game comes from experimenting with patterns you devise and seeing what happens. For instance, a straight line five units long goes through a number of permutations and eventually turns into four lines that flip back and forth between vertical and horizontal. U-shaped patterns of four or five units on a side are also interesting. Try it out!

The Program Listing

```
10 '
20 ' LIFE
30 '
40 DEFINT A-Z
50 INPUT"Screen Size"; M,N:DIM A(1,M+1,N+1)
60 PRINT"Move Cursor with Cursor Keys,"
70 PRINT"Type 1 for cell, 0 to erase cell,"
80 FOR Q=1 TO 4000 : NEXT Q
90 G=1:I=1:J=1:K=0:L=1:X=1:Y=1:A$="1"
100 CLS:LINE(0,0)-(4*M+7,4*N+7),1,BF
110 LINE (4,4)-(4*M+3,4*N+3),0,BF
120 '
130 ' Enter First Generation
140 '
150 LINE (4*I,4*J)-(4*I+3,4*J+3),VAL(A$),BF
160 A$=INKEY$:IF A$="" THEN 160
170 IF A$="1" OR A$="L" THEN 320
180 IF A$="1" OR A$="0" THEN A(0,I,J)=VAL(A$):GOTO 150
190 IF ASC(A$)=28 THEN I=I+1:IF I>M THEN I=M
200 IF ASC(A$)=29 THEN I=I-1:IF I<1 THEN I=1
210 IF ASC(A$)=30 THEN J=J-1:IF J<1 THEN J=1
220 IF ASC(A$)=31 THEN J=J+1:IF J>N THEN J=N
230 LINE(4*X,4*Y)-(4*X+3,4*Y+3),A(0,X,Y),BF
240 X=I:Y=J:A$="1":GOTO 150
250 FOR I=1 TO M
260 FOR J=1 TO N
270 LINE (4*I,4*J)-(4*I+3,4*J+3),A(K,I,J),BF
280 NEXT J:NEXT I
290 '
300 ' Calculate and Print Next Generation
310 '
320 SOUND 8000,1:PRINT @280,"Generation ";G;
330 FOR I=1 TO M
340 FOR J=1 TO N
350 S=A(K,I-1,J-1)+A(K,I,J-1)+A(K,I+1,J-1)+A(K,I-1,J)+
    A(K,I+1,J)+A(K,I-1,J+1)+A(K,I,J+1)+A(K,I+1,J+1)
360 IF A(K,I,J)<>0 THEN 390
370 IF S<>3 THEN A(L,I,J)=0 ELSE A(L,I,J)=1:
    LINE(4*I,4*J)-(4*I+3,4*J+3),1,BF
380 GOTO 400
390 IF S=2 OR S=3 THEN A(L,I,J)=1 ELSE A(L,I,J)=0:
    LINE(4*I,4*J)-(4*I+3,4*J+3),0,BF
400 NEXT J:NEXT I
410 K=1-K:L=1-L:G=G+1
420 GOTO 320
```

How the Program Works

Once it learns from the user how big a square is desired, the program (in line 50) sets up an array A(K,I,J). For each cell, K indicates current generation (K = 0) or next generation (K = 1). I and J are the vertical and horizontal coordinates, and the value of the array element itself indicates whether a particular cell is occupied or not: A(K,I,J) = 0 for unoccupied, and A(K,I,J) = 1 for occupied.

In lines 150 to 280 the user enters the starting configuration by moving the cursor keys and typing 1 or 0. This assigns 1 or 0 to the appropriate array elements.

Line 350 calculates the number of occupied cells surrounding a given cell. It does this by simply adding up the values in the eight adjacent cells. The resulting number determines whether the given cell will be born, live, or die. Lines 360 and 370 determine whether the square will be occupied on the next generation, and print out the appropriate changes in the screen display.

7

Learning Games

MULT

Dueling Digits

Multiplication Game

What happens when you ask your children how much it will cost to buy nine packages of bubble gum at five cents each? Do they sit there with glazed looks on their faces, trying to count on their fingers, or look about furtively for the nearest calculator? If so, the problem may be that they don't know their multiplication tables.

Maybe you can remember when you learned your times tables. If you're like most people, you found the experience tedious, difficult, and maybe even downright painful. Lots of people began to hate math just because of multiplication. You might even think that, with the advent of calculators, today's children don't *need* to know how to multiply — why not let the machine do it? But a moment's reflection tells you that kids still need to be able to do simple arithmetic, including multiplication, in their heads — not only to get through school, but so they won't have to reach for their calculators to answer the question posed above about bubble gum.

So learning the times tables is necessary. The question is how can it be made less painful?

Here's a program that turns learning your times tables into an exciting game. When the game starts, two numbers appear on the top line of the screen, one on the left and one on the right. They start moving toward each

other, bent on a fatal collision. If you can enter the product of the numbers (the "product" is what you get when you multiply two numbers together) before they crash into each other, you win. Otherwise, SMASH! You lose.

When you lose, the next two numbers appear one line lower down on the screen. As long as your answers are right, the numbers stay on the same level, but as soon as you miss one, either by entering the wrong answer or by not answering fast enough, you drop down a level, and, at the same time, the program speeds up. When you've lost six times, you're at the bottom of the screen, the game is over, and the program displays your score, the number of correct answers you've achieved.

Don't forget to press ENTER after you type the number. The program doesn't know in advance how many digits you want to type, so you need to tell it when you're done. For convenience, it's also possible to press the space bar to enter the number.

The Program Listing

```
10 '
20 ' MULT
30 '
40 T$=RIGHT$(TIME$,2)
50 FOR I=0 TO VAL(T$):R=RND(1):NEXT I
60 DEFINT A-Z:DIM N(40)
70 S=0:K=0:ST=100
80 '
90 ' Generate Operands & Print Positions
100 '
110 P1=44+K:P2=79+K
120 M1=INT(RND(1)*10)
130 M2=INT(RND(1)*10):MM=M1*M2
140 CLS:D=1:PD=82
150 P1=P1+1:P2=P2-1
160 IF ST=0 THEN 420
170 FOR I=1 TO ST:NEXT I
180 PRINT@P1,M1;:PRINT@P2,M2;
190 IF P1>=P2 THEN SOUND 16000,20:GOTO 400
200 '
210 ' Check Keyboard for Input
220 '
230 PRINT@80,"?";
240 A$=INKEY$:IF A$="" THEN 150
250 IF A$=CHR$(13) OR A$=CHR$(32) THEN 280
260 IF A$=CHR$(27) THEN PRINT@82,"   ":D=1:PD=82:GOTO 240
270 D=D+1:PD=PD+1:N(D)=VAL(A$):PRINT@PD,A$:GOTO 240
280 NN=0
290 '
300 ' Compute & Check Answer
310 '
320 FOR I=1 TO D
330 NN=NN+N(I)*10^(D-I)
340 NEXT I
350 IF NN<>MM THEN 400
360 SOUND 8000,10
370 S=S+1
380 FOR I=1 TO 150:A$=INKEY$:NEXT I
390 ST=ST-1:GOTO 110
400 SOUND 16000,10:K=K+40
410 IF K<240 THEN 380
420 CLS:PRINT@160,"Game Over,  Score =";S;
430 FOR I=1 TO 30:A$=INKEY$:NEXT I
440 A$=INKEY$:IFA$=""THEN 440
450 GOTO 70
```

How the Program Works

The first part of the program, lines 10 to 70, initializes variables and "seeds" the random number generator (so that the next random numbers generated won't always be the same) and initializes various parameters.

The next part of the program generates the two numbers that will race across the screen, and figures out where they will go. M1 and M2 are the numbers that are generated using the RND function in lines 120 and 130. P1 and P2 are the print positions of these two numbers, and are calculated in lines 110 and 150. The numbers are printed out using the PRINT@ statement in line 180.

Since the program has to move the numbers across the screen continuously and wait for keyboard input at the same time, it can't use the INPUT statement to read the keyboard. INKEY$ is called for here, in line 240. If INKEY$ finds that nothing has been typed, it goes back to line 150 to generate new positions for the numbers. But if INKEY$ finds that a digit has been typed, it stores it in array N. Then the program goes back to look for the next character from the keyboard. If the next character is another digit, the program must again save it in N. However, if the character is an ENTER or a space, the program goes to line 280 to figure out the complete number. This number is the combination of several digits: if the user typed a "2" and then a "7", the program must put them together to make 27.

The number of digits that have been typed in is recorded in variable D. The program now takes the first digit in N, multiplies it by 10, and adds the second digit. If there is another digit, the resulting number is again multiplied by 10, and the third digit added to the new result. This process is carried out in the loop in lines 320 to 340. The answer is recorded in variable NN. Now that the program knows what number the user typed in, it can compare it with the value it calculated itself in line 130 and recorded as MM. It does this in line 350. If the answer is right, control goes back to line 110 to move new numbers onto the screen.

If the answer is wrong, control goes to line 400. The program keeps track of how many lines down on the screen it is with the variable K. Now it can check this variable to see if the game is over. If so, the "Game over" message and the score are printed out in line 400. Otherwise, following the tone and a pause, control goes back to 110.

MATH

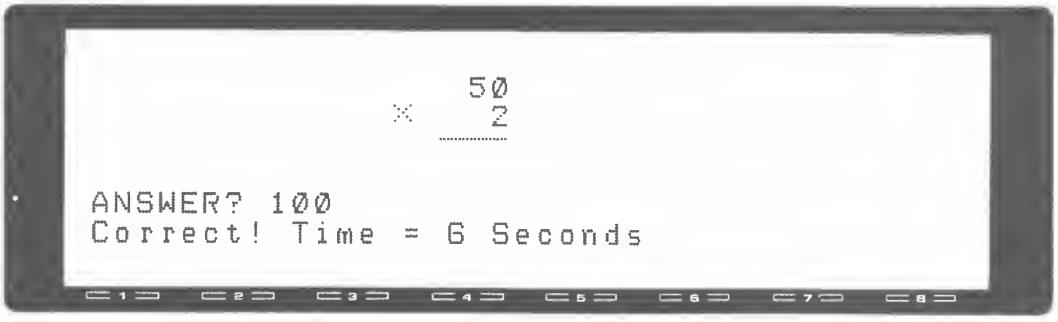
Mathomania

Arithmetic Practice Game

This is a real back-to-basics program. If you think that overuse of calculators and computers is causing your children (or you!) to forget fundamental arithmetic, then this program is for you. In fact, it serves the same function as the calculator-like arithmetic drill devices that are designed specifically to provide practice in arithmetic.

Mathomania provides practice in all four arithmetic operations: addition, subtraction, multiplication, and division. Also, it provides three levels of difficulty, ranging from easy to quite challenging.

When you RUN this program, it first asks you to enter the arithmetic operation you want to practice. Then it asks you for the difficulty level. When you've answered these questions, it proceeds to generate an arithmetic problem to your specifications, using random numbers. If your answer is wrong, you are informed of this fact, and given the correct answer. If your answer is correct, you are congratulated and the time you took to answer is displayed. Displaying the elapsed time keeps the interest high, and a game may be made out of the learning process: two or more people may compete to see who can do the problems fastest.



Once a problem is completed, you can start the next one by pressing any key, including ENTER. This displays the new problem and starts the clock running.

When you want to change to a new arithmetic operation or a new difficulty level, pressing the letter "c" starts the program over.

Program Listing

```
10 '
20 ' MATH
30 '
40 T$=RIGHT$(TIME$,2)
50 FOR I=0 TO VAL(T$):R=RND(1):NEXT I
60 DIMX(1)
70 CLS:PRINT@50,"1. ADDITION";:PRINT@90,"2. SUBTRACTION";:
   PRINT@130,"3. MULTIPLICATION";:PRINT@170,"4. DIVISION"
80 PRINT:INPUT "CHOOSE 1, 2, 3 OR 4";F
90 INPUT "LEVEL (1,2 or 3)";L
100 D=10^L-1
110 CLS:P=55
120 ON F GOTO 160,230,320,320
130 '
140 ' Addition Routine
150 '
160 FOR I=0TO1:GOSUB 700:GOSUB 710
170 PRINT@Q(I),X(I):P=P+40
180 NEXT I
190 PRINT@94,"+":RA=X(0)+X(1):GOTO 560
200 '
210 ' Subtraction Routine
220 '
230 FORI=0 TO 1:GOSUB700:NEXT
240 IF X(0)<X(1)THENT=X(0):X(0)=X(1):X(1)=T
250 FOR I=0 TO 1
260 GOSUB 710:PRINT@Q(I),X(I):P=P+40
270 NEXT I
280 PRINT@94,"-":RA=X(0)-X(1):GOTO 560
290 '
300 ' Multiplication Routine
310 '
320 ON L GOTO 330,360,390
330 D=9
340 FOR I=0 TO 1:GOSUB 700:NEXT I
350 GOTO 410
360 D=99:I=0:GOSUB 700
370 D=9:I=1:GOSUB 700
380 GOTO 410
390 D=999:I=0:GOSUB 700
400 D=99:I=1:GOSUB 700
410 IF F=4 THEN 490
420 IF X(0)<X(1) THEN T=X(0):X(0)=X(1):X(1)=T
```

```
430 FOR I=0 TO 1:GOSUB 710
440 PRINT@Q(I),X(I):P=P+40:NEXT I
450 PRINT@94,"x":RA=X(0)*X(1):GOTO 560
460 '
470 ' Division Routine
480 '
490 PD=X(0)*X(1):RA=X(0):X(0)=PD
500 FOR I=0 TO 1:GOSUB 710
510 PRINT@Q(I),X(I);:P=P+40:NEXT I
520 PRINT@94,"/";:GOTO 560
530 '
540 ' Input and Check Answer
550 '
560 PRINT@136,CHR$(241);CHR$(241);CHR$(241)
570 T=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))
580 PRINT:INPUT"ANSWER";A
590 T=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))-T:IF T<0 THEN T=T+3600
600 IF A<>RA THEN 640
610 PRINT "Correct! Time =";T;"Seconds";
620 A$=INKEY$:IFA$=""THEN620
630 IF A$="c" THEN 70 ELSE 110
640 PRINT "Incorrect!  The Correct Answer is";RA
650 A$=INKEY$:IFA$=""THEN650
660 IF A$="c" THEN 70 ELSE 110
670 '
680 ' Generate Operands and Compute Print Positions
690 '
700 X(I)=INT(RND(1)*D)+1:RETURN
710 IF X(I)<10 THEN Q(I)=P+2:RETURN
720 IF X(I)<100 THEN Q(I)=P+1ELSE Q(I)=P
730 RETURN
```

How the Program Works

The program starts off (lines 10-70) by asking the user for a mathematical function (F = 1 for addition, F = 2 for subtraction, and so on). Then it asks (lines 90, 100) for the difficulty level, from 1 to 3, and assigns this number to variable L. This variable is then used to raise 10 to a power; the resulting new variable D has a 3-digit value for hard problems, a 2-digit value for medium, and a 1-digit value for the easy ones. This is used in the subroutine at line 700 where the random numbers are generated.

Each of the routines (addition, subtraction, and so on) generates two random numbers — X(0) and X(1). The difficulty level, L, determines how large these numbers will be. For example, in the multiplication routine, if L = 2 we go to line 360. Since D represents the upper limit of the number, it is set to 99. Then the subroutine at 700 is called, which generates a random number between 0 and 99, and also determines the print position depending on the size of the number in 710 and 720. The second multiplier (line 370) will be between 0 and 9.

The division routine operates slightly differently. The divisor and the quotient (the answer) are the two numbers generated. The dividend is then generated by multiplying these two numbers together. This is done to ensure that all answers will come out in whole numbers.

TWORD

Typerony

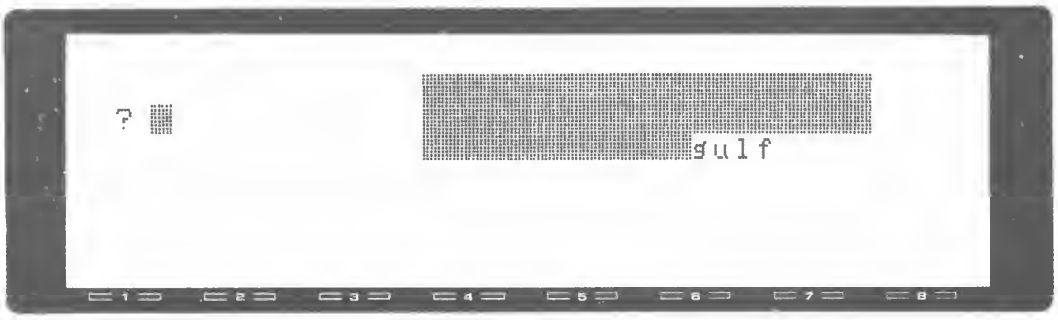
Word Typing Game

Typerony is a game whose purpose is to improve your typing skill. You can use it to teach yourself touch-typing, or perhaps to unlearn the slow "hunt and peck" technique you've been trying to get by with all these years.

The program flashes a word on the screen, and you type the same word as fast and accurately as you can. Your goal in the game is to complete the filling in of a rectangular box on the screen in as short a time as possible. Every time you type a correct word, more of the box is filled in. If you type a word incorrectly, part of the box is erased, so that you lose ground. Thus, you are motivated to type accurately as well as fast. When you have typed twenty words correctly, the box is completed, and the time you took is displayed.

Because the computer keeps track of your elapsed time, the game can be played with two people, who compete with each other to see who can complete the box in the shortest time. Or, individuals can compete against themselves.

Typerony has three levels of difficulty, from which you select at the beginning of the game. The higher the difficulty level, the longer the word you are asked to type.



The words used in the program have been selected to improve your typing skills by exercising all the finger positions. It is possible to add new words to those already stored in the program, or to substitute different words. We show you how to do this later, in the section "How the Program Works".

Program Listing

```
10 '
20 ' TWORD
30 '
40 DIM W$(2,40)
50 REM data statements
60 DATA you,for,win,box,can,try,ink,now,ask,him,her,ago,but,the,
   our,air,hip,cab,wet,pin,oil,joy,hum,gas,rag,six,off,and,was,
   bed,bug,old,egg,may,big,did,had,kid,led,run
70 DATA done,does,give,true,lend,they,back,sigh,paid,when,dark,
   what,copy,wish,sure,best,very,exit,quit,zero,must,show,feel,
   fair,boil,such,navy,kind,play,milk,joke,gulf,many,lawn,your,
   went,hand,camp,peek,sing
80 DATA facts,while,women,until,after,where,brick,quiet,first,
   power,three,state,aches,ruler,tough,favor,onion,brave,verbs,
   jelly,judge,flame,heard,thank,equal,reply,dusty,throw,extra,
   happy,bench,would,about,being,above,check,every,fifty,eight,offer
90 '
100 ' Read in Word Lists & Initialize
110 '
120 FOR L=0 TO 2
130 FOR I=1 TO 40
140 READ W$(L,I):NEXT I
150 NEXT L
160 T$=RIGHT$(TIME$,2)
170 FOR I=0 TO VAL(T$):R=RND(1):NEXT I
180 CLS:INPUT "Choose Level 0, 1 or 2";L
190 CLS:PP=55:P=0
200 T=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))
210 '
220 ' Generate & Print Word
230 '
240 J=INT(RND(1)*40)+1:IF W$(L,J)=" " THEN 240
250 WD$=W$(L,J)
260 PRINT@PP+P,WD$
270 PRINT@81,"        ";:PRINT@80," ";:INPUT WI$
280 IF WI$=WD$ THEN 390
290 '
300 ' Wrong Answer
310 '
320 PRINT@PP+P,"         ";
330 P=P-4:IF P<0 THEN P=16:PP=PP-40
340 IF PP+P=15 THEN 510
```

```
350 PRINT@PP+P,WD$;" ";:GOTO 270
360 '
370 ' Right Answer
380 '
390 PRINT@PP+P,CHR$(239);CHR$(239);CHR$(239);CHR$(239);
400 P=P+4
410 IF P=20 THEN PRINT@PP+P,"     ";:P=0:PP=PP+40
420 IF PP<>255 THEN GOTO 240
430 '
440 ' End of Game Routine
450 '
460 PRINT@240,"Finished!"
470 T2=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))
480 T2=T2-T:IF T2<0 THEN T2=T2+3600
490 PRINT@251,"Time =";T2;" Seconds";
500 SOUND 4000,20:GOTO 530
510 PRINT@80,"You Lose! Press Key to Play Again";
520 SOUND 16000,20
530 A$=INKEY$:IF A$=""THEN 530
540 GOTO 180
```

How the Program Works

As you can see from the listing, this program stores the typing-test words in DATA statements (lines 60, 70, and 80). An array is set up which is 40 elements long and three elements wide. The values of all 120 words (40 easy, 40 medium, and 40 hard) are then inserted into the array using the nested FOR...NEXT loops in lines 120 to 150.

The program then asks you (line 180) to choose a difficulty level: 0, 1, or 2. This level and a random number J (generated in line 240) are then used to extract one of the words from the array, in line 250. This word is then displayed, and the user is asked to type in the same word in line 270.

If the word typed in matches that in the array (as determined in line 280), the program branches to line 350, where the box is filled in and the program goes back to line 240 for a new word. If the word typed in does not match, then part of the box is overprinted with spaces (line 300), and the program returns to line 270 to display the same word again.

When all forty words have been completed, the program goes to line 420 to print out the "finished!" message and display the elapsed time.

To change the words used in the program, you first need to change the DATA statements in lines 60, 70, and 80. If you use the same number of words, forty, in each list, this is the only change you will need to make. Simply retype the DATA statements (making sure to use exactly forty words). If you want to add additional words, you will also need to change the index that tells the program how many items there are per list. This is the number 40 in lines 40, 130, and 240. For instance, if you want to use 100 items in each of the three lists, you would change these three lines to those shown below, substituting 100 for 40 in each:

```
40 DIM W$(2,100)

130 FOR I = 1 TO 100

240 J=INT(RND(1)*100)+1:IF W$(L,J)=" " THEN 240
```

HANG

Electric Hangman

Classic Spelling Game

Remember the old word-guessing game you used to play where you had to guess a word, and every time you missed a letter another line was added to the figure of a hanging man? Electric Hangman is a space-age version of the old hangman game, brought up-to-date with automatic scoring and computer-generated graphics. It's great for kids or for anyone who enjoys games, and is perfect for passing the time on long trips. As you play, it will also improve your vocabulary!

Electric Hangman is a two-person game. One person thinks of a word and types it into the computer, without the second person seeing what it is. The second person tries to guess what it is, by trying individual letters. The computer remembers the word the first player thinks of, draws the picture of the hanged man, and tells the second player if he's guessed right or wrong. It keeps track of the letters guessed correctly and shows them in their proper places, and also displays the incorrect guesses. Finally, the program informs you if you've won or lost.

Words can be any length, so the game will work with a variety of age levels. But no matter how long the word is, the guesser can make only seven wrong guesses — if he doesn't know the word by then, he loses.

When you first run Electric Hangman, it asks, "What is your word?" The first player must then think of a word, up to sixteen letters long, and type it in, pressing [ENTER] when finished. Make sure the [CAPS LOCK] key is not toggled to capitals, since the program only recognizes lower case letters. Your word should not contain any capital letters, spaces, or punctuation, including hyphens or numbers. If you make a mistake typing in the word, you can change it with the backspace key. Of course, the first player should type in this word without the second player seeing it.

Once the word is entered, the program erases the word. Then it draws a picture of the gallows and a series of dashes to represent the letters of the word. Thus, the second player knows at the beginning how many letters there will be in the word, but not what they are. The program then displays the word "GUESS". The second player types a letter. (Again, remember it should be lower case.) If the letter is in the word, the program plays a few victorious trumpet notes and fills in the letter above the appropriate dash, so that the second player can tell where it occurs in the word. If it is not correct, the program adds the next body part to the hanged man: head, arm, torso, or leg. Then it prints out the incorrect letter, following the word "Wrong", and plays an appropriate dirge.

The second player continues to guess letters. If he guesses the word in less than seven tries, the program plays a fanfare and proclaims "You Win!" But if the player exceeds seven tries, the program says "You Lose", and reveals the word.

To start the game over, press any key.

Program Listing

```
10 '
20 ' HANG
30 '
40 DIM CX(25)
50 CLS:INPUT "WHAT IS YOUR WORD";W$
60 L=LEN(W$):P=288:G=145:M=0:C=0:CLS
70 '
80 ' Draw Gallows, Print Blanks, get guess-letter
90 '
100 LINE (2,60)-(32,60)
110 LINE (17,60)-(17,1)
120 LINE (17,1)-(47,1)
130 LINE (47,1)-(47,11)
140 FOR I=1 TO L
150 IF MID$(W$,I,1)<>" " THEN B$="q" ELSE B$=" ":C=C+1
160 PRINT@(P+2*I),B$;:NEXT I
170 PRINT @60,"GUESS:  "
180 A$=INKEY$:IF A$="" THEN 180
190 A=ASC(A$)-97: IF A<0 OR A>25 THEN SOUND 12000,50: GOTO 170
200 IF CX(A)<>0 THEN SOUND 12000,50: GOTO160 ELSE CX(A)=1
210 K=0: PRINT @67,A$;
220 '
230 ' Check Guess Against Word
240 '
250 FOR I=1 TO L
260 IF A$=MID$(W$,I,1) THEN PRINT@(P-40+2*I),A$;:K=1:C=C+1:
    SOUND 4697,10:SOUND 3516,30
270 NEXT I
280 IF C<>L THEN 320
290 PRINT@ 140,"You Win!            ";
300 SOUND 0,20:SOUND 7032,20:SOUND 5586,10:SOUND 5586,30:
    SOUND 4697,20:SOUND 4697,20:SOUND 3516,10:SOUND 3516,30
310 GOTO 570
320 IF K<>0 THEN 170
330 M=M+1
340 PRINT@140,"Wrong:";
350 G=G+2:PRINT@G,A$;
360 SOUND 9394,30:SOUND 8368,10:SOUND 7900,20:SOUND 9394,20
370 '
380 ' Routine to Draw Body Parts
390 '
400 ON M GOTO 410,450,460,470,480,490,500
410 LINE(46,12)-(48,12):PSET(49,13)
420 LINE(50,14)-(50,16):PSET(49,17)
430 LINE(46,18)-(48,18):PSET(45,17)
```

```
440 LINE(44,14)-(44,16):PSET(45,13):GOTO 170
450 LINE(47,19)-(47,21):GOTO 170
460 LINE(46,22)-(39,29):GOTO 170
470 LINE(48,22)-(55,29):GOTO 170
480 LINE(47,22)-(47,33):GOTO 170
490 LINE(46,34)-(39,41):GOTO 170
500 LINE(48,34)-(55,41)
510 '
520 ' End of Game
530 '
540 PRINT @133,"You lose: the word is      ";
550 PRINT@ 180,W$;
560 SOUND 0,30:SOUND 9394,30:SOUND 8368,10:SOUND 7900,20:
    SOUND 9394,20:SOUND 6642,40
570 PRINT@60,"        ";
580 A$=INKEY$:IF A$="" THEN 580
590 CLEAR: GOTO 50
```

How the Program Works

The word to be guessed is assigned to variable W$ in the INPUT statement in line 50. The program then draws the gallows and prints the row of blanks (lines 100 to 160). In line 180 the program reads in the letter guessed by the second player. This character is assigned to variable A$. Lines 190 and 200 then check to be sure the letter is lower case and that it has not been typed before. If either of these conditions occur, a tone sounds. To make sure that the same letter isn't guessed twice, the program maintains an array CX. Elements of this array are set to 1 whenever the corresponding letter, from 0 to 25, is guessed.

If the letter passes these tests, the program, in lines 250 to 270, checks it against all the letters in the word W$. If a match is found, two notes are played. If there is no match, then the program branches to line 320. Here the word "Wrong" is printed on the screen, the letter is added to the list of wrong letters, and the melody in line 360 is played.

The variable C (see line 280) contains the total number of correct guesses. When C is equal to L, the length of the word originally entered, the program goes from line 280 on to line 290 where it prints out "You Win!" and plays the fanfare (line 300).

Line 260 sets the variable K to 1 if a match is found, and to 0 if there is no match. In line 320 the program decides, on the basis of whether K is 1 or 0, whether to go to 170 for another guess, or to print "Wrong", play the dirge for the wrong letter, and draw the appropriate body part (lines 400 to 500).

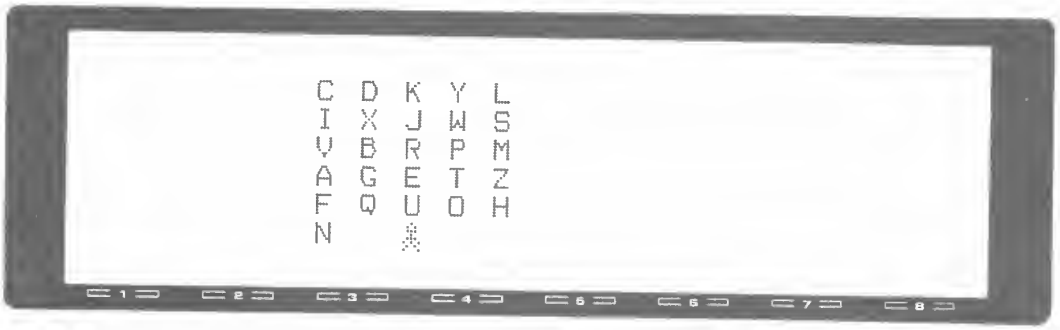
ALPHA

Alphabet Soup

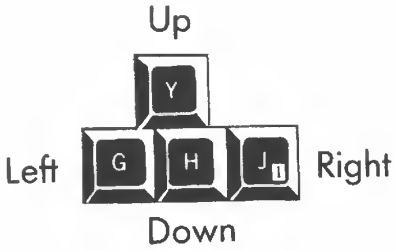
Fast Paced Alphabet Game

This program provides fun for the whole family, from young children to adults who want to practice their hand-eye coordination. However, it is especially appropriate for children who are just learning the alphabet. It teaches the alphabet as well as letter recognition, using both sight and sound to reinforce the learning process. Thus, it is a valuable tool for the *Sesame Street* age level. Kids can play the game by themselves, or two or more can compete against each other, to see who can finish the game fastest — each player's time is displayed after his or her turn.

At the start of the game all the letters of the alphabet are displayed, in random order, in a square matrix on the screen (actually a 5-by-5 square with an extra letter at the bottom, since there are 26 letters). A little puppet appears at the bottom of the screen, and your job is to move the puppet from "A", then to "B", and so on through the alphabet to "Z". Since the letters are arranged in random order, you must quickly find each letter and figure out the fastest way to move the cursor to it. Each time you land on a letter, one note of the "Alphabet Song" is played, so that if you move fast enough the notes blend together into a recognizable melody.



Four letter keys are used to move the puppet because they fall more naturally under the fingers than do the regular cursor keys:



Once the puppet is positioned over the correct letter, you press the letter "S" to gobble up the letter. It works well to use your right hand for moving the puppet, and your left for gobbling the letters.

The object of the game is to gobble all 26 letters, in order, as fast as possible, until the screen is completely blank. If you get a single letter out of order, you lose, and the game is over. At the end of the game the total time you took to gobble all 26 letters is displayed, so you can tell how well you've done. You'll find that as you become familiar with the keys you will be able to speed up, improve your final time and make the "Alphabet Song" play faster and faster.

To start the game over, simply press the space bar.

Program Listing

```
10 '
20 ' ALPHA
30 '
40 DEFINT A-Z:DIM A(25):DIM B(4,5):DIM S(25)
50 FOR I=0TO25:S(I)=4697:NEXT
60 M$=CHR$(147):T$=RIGHT$(TIME$,2)
70 FOR I=0 TO VAL(T$):Z=RND(1):NEXT I
80 '
90 ' Assign Notes to Letters
100 '
110 A(1)=4697:A(2)=4184:A(3)=3728:A(4)=3516:A(5)=3134:A(6)=2793
120 X$="11556654433222215544325432"
130 FOR I=0TO25:R=VAL(MID$(X$,I+1,1)):S(I)=A(R):NEXTI
140 '
150 ' Randomize Letters
160 '
170 FOR I=0 TO 25:A(I)=I:NEXT I
180 FOR I=1TO4:B(I,5)=-33:NEXT I
190 FOR J=0 TO 25
200 R=INT(RND(1)*26)
210 T=A(J):A(J)=A(R):A(R)=T
220 NEXT J
230 '
240 ' Print Letters
250 '
260 CLS:Y=0:X=-1
270 FOR I=0 TO 25
280 X=X+1:IF X>4 THEN X=0:Y=Y+1
290 P=50+2*X+40*Y:B(X,Y)=A(I)
300 PRINT@P,CHR$(B(X,Y)+65)
310 NEXT I
320 X=2:Y=5:P=254:L=-1
330 PRINT@P,M$:PP=P:CP$=" "
340 SOUND S(1),20
350 TM=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))
360 '
370 ' Move Man Via Key Input
380 '
390 A$=INKEY$:IFA$="" THEN 390
400 IF A$="s" THEN 510
410 IF A$="y" THEN Y=Y-1:IF Y<0 THEN Y=0
420 IF A$="h" THEN Y=Y+1:IF Y>5 THEN Y=5
430 IF A$="g" THEN X=X-1:IF X<0 THEN X=0
440 IF A$="j" THEN X=X+1:IF X>4 THEN X=4
450 P=50+2*X+40*Y:PRINT@P,M$:IFPP=PTHEN390
```

```
460 PRINT@PP,CP$:CP$=CHR$(B(X,Y)+65)
470 PP=P:GOTO 390
480 '
490 ' Check for Correct Hit
500 '
510 L=L+1:IF B(X,Y)<>L THEN PRINT@65,"You Lose!":
    SOUND12538,40:GOTO 590
520 SOUND S(L),5:CP$=" ":B(X,Y)=-33
530 IF L<25 THEN 390
540 TE=VAL(RIGHT$(TIME$,2))+60*VAL(MID$(TIME$,4,2))-TM:
    IF TE<0 THEN TE=TE+3600
550 PRINT@61,"You Win!";
560 PRINT@101,"Time = ";TE;"Seconds";
570 SOUND0,20:FOR I=0TO6:SOUNDS(I),20:NEXT I
580 SOUND 0,20:FORI=7TO12:SOUNDS(I),20:NEXTI:SOUNDS(1),20
590 A$=INKEY$:IF A$="" THEN 590
600 GOTO 170
```

How the Program Works

The first part of the program (lines 40 to 70) simply initializes various arrays and constants. The next part (90 to 130) generates the notes of the alphabet song, putting the values of the notes, one for each letter, into the array S. The appropriate note for each letter will then be played later in the program (line 520), every time a correct letter is typed.

An interesting feature of this program is the randomization routine. To set up the array of letters we can't simply generate 26 letters of the alphabet, one after the other, since this would generate some letters more than once and skip others. What we need to do is start with all 26 letters, and then "shuffle" them, rather like a deck of cards, before we write them onto the screen.

This is accomplished in lines 150 to 220. This section first sets up an array A, which contains the numbers from 0 to 25, each one representing a letter of the alphabet. That is, A(0) = 0, A(1) = 1, and so forth. Line 200 then generates a random number in the range 0 to 25. This is used to swap two of the elements on the list: the first one, A(0), and the one whose index is the random number. For instance, if the random number is 7, then the first and seventh elements of the array will be exchanged. The second time, the second element, A(1), is swapped with a different random element, and so on, until every element in the array has been swapped with another element. This results in complete randomization of all 26 letters.

The section of the program from 240 to 350 prints the letters out on the screen in their newly-randomized order. Lines 370 to 470 then wait for the user to type one of the cursor-motion characters "y", "h", "g", or "j"; or to type "s" to erase the letter. If "s" is typed, the program branches to line 510, where it figures out whether the letter selected is, in fact, the correct letter of the alphabet. If it's the wrong letter, the program prints the "You Lose!" message and goes to wait (in line 590) for the user to press any character to continue the game. If the character is correct, the program plays the corresponding note of the alphabet song, and erases the letter from the screen (line 520). When all 26 letters have been removed from the screen in the correct order, the program prints the "You Win" message, calculates the elapsed time (line 540), and prints it out (line 560).

8

Practical Calculators

CALC

Easycalc

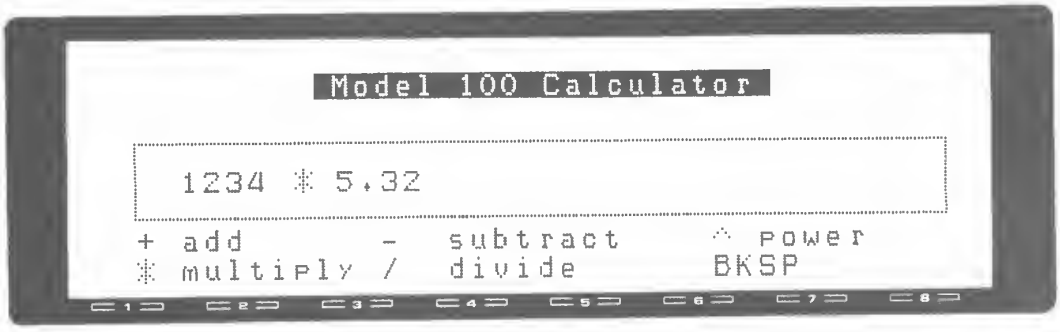
Five Function Calculator

You are the proud owner of one of the smallest and most powerful computers on the market. You can take it anywhere and execute sophisticated and astounding programs that amaze your friends and elicit envy from all quarters. The question is, can it add 2 and 2? The answer is that it can, but it's not very convenient. That is, not without this program.

You can always do arithmetic in BASIC, by writing a PRINT statement followed by the arithmetic expression you want to evaluate. However, this is an inelegant and inconvenient solution to the problem. Why can't your computer do simple arithmetic as easily as a calculator?

The purpose of this program is to turn your Model 100 into a super-accurate five-function calculator. (If you want a sophisticated, scientific RPN calculator, see the following program.) Your five-function calculator will do everything a typical calculator can, but with an added advantage — it calculates to an accuracy of fourteen decimal places, instead of the usual eight.

The first four functions are the usual addition, subtraction, multiplication, and division. The fifth function is raising a number to a power. This is a generalized version of the square root function found in many simple calculators. However, besides being able to find square roots, this program



will also find cube roots (or any other roots), and raise a number to any power (such as squaring it, cubing it, and so on).

Operation of Easycalc

When you run the program, you'll see a screen display that shows all the possible arithmetic operations. The "operations area" (the space where the numbers you type in will appear) will be outlined in the middle of the screen. To use the program you enter a number, followed by an arithmetic symbol like "+" or "/", then the second number, and finally an equal sign. The program will immediately print out your answer. You can use the backspace key if you make a mistake typing anything in.

The program will not let you make an invalid entry. For instance, if you type a letter instead of a digit, it will be ignored, as will an arithmetic symbol if you have not yet entered a number.

Once you have your answer, the display of available options shown on the bottom of the screen will change. You can elect to retain the answer for the next computation (called "chaining" on some calculators), you can clear the display for the next calculation, or you can end the program.

As you can see from the listing, it would not be hard to add additional functions to this routine. We'll say more about this in the description of the program's operation.

Program Listing

```
100 '  CALC
110 '
120 REM 5 FUNCTION CALCULATOR
130 CLS:CALL 17001: PRINT@50," Model 100 Calculator ":CALL 17006
140 A$=""
150 LINE (12,25)-(227,45),1,B: GOSUB 430
160 PRINT@ 242,"+ add       -  subtract     ^ power"
170 PRINT@ 282,"* multiply  /  divide       BKSP";
180 B$=""
190 X$= INKEY$:IF X$="" THEN 190
200 IF X$="+" OR X$="-" OR X$="*" OR X$="/" OR X$="^" THEN 260
210 IF ASC(X$)=8 AND LEN(A$)<=0 THEN 190 ELSE IF ASC(X$)=8
    THEN A$=LEFT$(A$,(LEN(A$)-1)):GOSUB 430:PRINT" ";A$:GOTO 190
220 '
230 'Get number
240 '
250 IF ASC(X$)<46 OR ASC(X$)>57 OR ASC(X$)=47 THEN 190
    ELSE A$=A$+X$:GOSUB 430:PRINT" ";A$:GOTO 190
260 A$=A$+" "+X$:IF LEN(A$)=2 GOTO 130
270 GOSUB 430:PRINT " ";A$+" ";
280 Y$=INKEY$:IF Y$="" THEN 280
290 IF ASC(Y$)=8 AND LEN(B$)<=0 THEN A$=LEFT$ (A$,(LEN(A$))-1):
    GOTO 190 ELSE IF ASC(Y$)=8 THEN B$=LEFT$(B$,(LEN(B$)-1)):
    GOSUB 430:PRINT" ";A$+" "+B$:GOTO 280
300 IF Y$="=" THEN 310 ELSE IF ASC(Y$)<46 OR ASC(Y$)>57 OR ASC(Y$)=47
    THEN 280 ELSE B$=B$+Y$:GOSUB 430:PRINT" "; A$+" ";+B$:GOTO 280
310 GOSUB 430:PRINT" ";A$+" ";B$+" = ";
320 '
330 'Compute and display answer
340 '
350 IF X$="+" THEN Z=VAL(A$)+VAL(B$):PRINT Z;:GOTO 400
360 IF X$="-" THEN Z=VAL(A$)-VAL(B$):PRINT Z;:GOTO 400
370 IF X$="*" THEN Z=VAL(A$)*VAL(B$):PRINT Z;:GOTO 400
380 IF X$="/" THEN Z=VAL(A$)/VAL(B$):PRINT Z;:GOTO 400
390 IF X$="^" THEN Z=VAL(A$)^VAL(B$):PRINT Z;
400 PRINT@240,"     'E' to end, 'R'etain value      ":
    PRINT"      or     'ENTER' to continue      ";
410 N$=INKEY$:IF N$="" THEN 410
420 IF ASC(N$)=13 THEN GOSUB 430:GOTO 130 ELSE IF N$="E" THEN
    END ELSE IF N$="R" THEN A$=STR$(Z):GOSUB 430: PRINT" ";A$:
    GOTO 160 ELSE GOTO 410
430 PRINT@164,STRING$(32,32):PRINT@163,"";:RETURN
```

How the Program Works

This program goes to considerable trouble to protect the user of the program from making mistakes. For this reason, all entries are made in the form of strings, rather than numbers. The INKEY$ function is used for this purpose.

An unusual feature of the program is the use of the "reverse video" routine in line 130. This is a machine-language subroutine in ROM, which can be called with BASIC's CALL statement. It changes the screen display from black on white to white on black. A similar routine changes the display back to the normal mode. This is a cute trick to know when you want to emphasize something on the screen.

After the title is printed, line 150 draws the outline of the calculation area, and lines 160 and 170 add the display prompts at the bottom of the screen. Line 190 waits for a key to be pressed, and line 200 analyses it to see if it's a math symbol ("+", "/", etc.). Line 210 deals with the case where the character is a backspace; if there are already characters in the input string A$, one is removed. If not, the program returns for another character.

In line 250, if the character is a digit or a decimal point, it is added to the input string A$, and the program returns for another character. Line 260 deals with the error situation where the first character is a math symbol. If there is no error, the first number and the math symbol are printed out in line 270, and the program goes on to line 280 to wait for the second number to be entered. Line 290 is the same as line 210 but operates on the second number. The equal sign is checked for in line 300, and if it's not found, then a digit or decimal point is searched for.

Lines 350 to 390 compute and display the answer for the various math operations. Lines 400, 410, and 420 then print the new prompts and wait for the user's response: either "E" to end the program, "R" to retain the old answer, or ENTER to start over.

Adding New Functions

To add new functions to the program, you need to insert the appropriate formula following line 390. Simply follow the same format as the previous lines. One example might be to add a percent (%) function. In addition to inserting this line, line 200 must have the new symbol added so it will be recognized as valid by the program. If you want, you can also modify lines 160 and 170 so that the new symbol is shown in the screen display. This is nice for the user but not essential for the operation of the program.

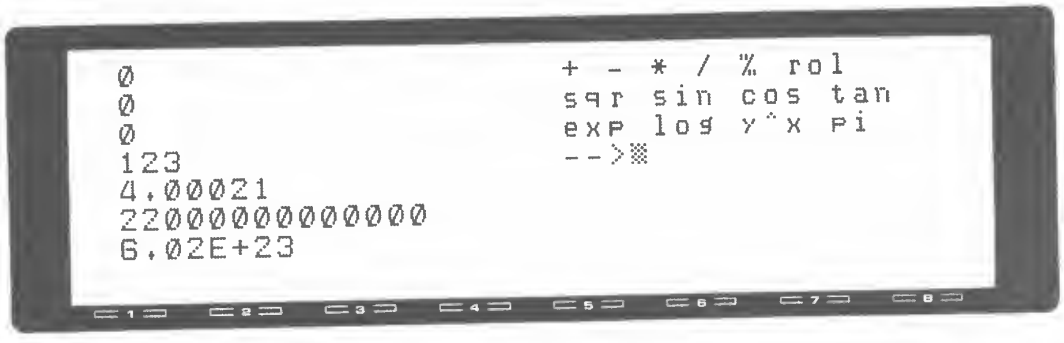
RPN

Sci-Calc

Scientific Calculator

If you've ever used an RPN calculator, you know how elegant and convenient this approach is compared with the "algebraic" approach used by other calculators. This program turns your Model 100 into a sophisticated, scientific RPN calculator. The advantage over hand-held RPN calculators is that you can see all the levels of the "stack" on the screen at once. (We'll explain what the "stack" is in a moment.) Being able to see the entire stack not only clarifies its operation while you're using the calculator, but also makes the Model 100 a perfect teaching tool for those learning how to use RPN calculators.

What's the secret of RPN calculators? "RPN" stands for "Reverse Polish Notation". It's a scheme invented by computer scientists for representing algebraic expressions. Suppose you want to add 2 and 3. In normal algebraic form this would be represented as "2 + 3", and this is how you would enter it on an algebraic calculator. In RPN you would change this around slightly, and say "2, 3, +". That is, in RPN you start out with two numbers and then enter the arithmetic operator; whereas, in algebraic mode you type one number, then the operator, and then a second number.



What's the advantage of RPN? Not much in one-operation problems like 2 + 3. But when you have a larger formula like:

x = ((44 + 12.94) * (9.2-33) + 4) / 4

then RPN can offer significant advantages. One is that you can see all the intermediate results as you go along. If you have a feeling for the quantities involved, this can help avoid mistakes, since you can see immediately if a number is out of the ballpark.

The secret to an RPN calculator is that intermediate results are saved on a "stack", where they are available for subsequent parts of the calculation. This stack is simply a series of places where numbers can be stored. When you put a number on the bottom of the stack, all the numbers which were on the stack before move up one space. When you take a number off the bottom, all the other numbers move down. This sort of stack is called LIFO, for "Last In First Out", meaning that the last thing you put on the stack is the first thing you get off. In hand-held calculators you can only see the item on the bottom of the stack, so it's hard to get a feel for what's happening. In our RPN program, all the stack locations are shown, and the operation of the stack is much clearer.

Our RPN calculator has other advantages over hand-held calculators — it calculates to fourteen places of accuracy, and you can enter numbers directly in scientific notation, such as 3.3E-10.

Operating the Program

When you first start the RPN program, you'll see the stack displayed on the left of the screen: a column of seven numbers, all set to zero when you start. On the right will be the various operations which the program allows: +, -, *, /, %, ROL, SQR, SIN, COS, EXP, LOG, Y^X, and PI. There is also an arrow and a flashing cursor to show where your input will go.

Let's try a simple example: adding the numbers 44 and 100. Type the number 44, then press ENTER. You'll see the number appear on the bottom of the stack. Now type the number 100, and again press ENTER. Now you'll see both numbers on the stack, the 44 just above the 100. Finally type the plus sign, "+", and again press ENTER. The answer, 144, will now appear on the bottom of the stack; the other numbers will be gone.

Want to find the square root of the result? Simply type "sqr" and press ENTER. The square root of 144, which is 12, will appear at the bottom of the stack and the 144 will be gone. The other functions can be performed in the same way.

The real power of the RPN approach is in solving more complex expressions. Let's solve the expression used earlier as an example:

x = ((44 + 12.94) * (9.2-33) + 4) / 4

To solve such expressions in RPN, you start inside the deepest level of parentheses, and then work outward, always saving intermediate results on the stack.

Enter the number 44. That is, put the number on the bottom of the stack, by typing "44" and then pressing ENTER. Then, enter 12.94, then add these two numbers by entering the plus sign "+". The two numbers will be added, and the intermediate result, 56.94, will be displayed on the bottom of the stack. The original numbers disappear. Now enter 9.2, then 33, then subtract them by entering the minus sign. Now you'll have two intermediate results on the stack: -23.8 just below 56.94. To multiply these results, all we need to do is enter the times sign "*", which leaves −1355.172 on the stack. We add 4 to the result by entering 4 and a plus sign, then divide by 4 by entering 4 and the divide sign "/", to yield the final result −337.793.

Program Listing

```
100 'RPN
110 '
120 DIM ST(8)
130 '
140 'print stack contents, get input
150 '
160 CLS : FOR I=6 TO 0 STEP -1
170 PRINT ST(I)
180 NEXT I
190 PRINT@21,"+ - * / % rol"
200 PRINT@61,"sqr sin cos tan"
210 PRINT@101,"exp log y^x pi"
220 PRINT@141,"";:LINE INPUT"-->";A$
230 '
240 'check for arithmetic operation
250 '
260 IF A$="+"THEN ST(1)=ST(1)+ST(0): GOTO 530
270 IF A$="-"THEN ST(1)=ST(1)-ST(0): GOTO 530
280 IF A$="*"THEN ST(1)=ST(1)*ST(0): GOTO 530
290 IF A$="/"THEN ST(1)=ST(1)/ST(0): GOTO 530
300 IF A$="%"THEN ST(1)=ST(1)*ST(0)/100: GOTO 530
310 '
320 'check for function operation
330 '
340 IF A$="rol" THEN 530
350 IF A$="sqr"THEN ST(0)=SQR(ST(0)): GOTO 160
360 IF A$="exp"THEN ST(0)=EXP(ST(0)): GOTO 160
370 IF A$="log"THEN ST(0)=LOG(ST(0)): GOTO 160
380 IF A$="sin"THEN ST(0)=SIN(ST(0)): GOTO 160
390 IF A$="cos"THEN ST(0)=COS(ST(0)): GOTO 160
400 IF A$="tan"THEN ST(0)=TAN(ST(0)): GOTO 160
410 IF A$="y^x"THEN ST(1)=ST(1)^ST(0): GOTO 530
420 IF A$="pi"THEN A$="3.1415926535898": GOTO 460
430 '
```

```
440 'move stack up, put number in stack
450 '
460 FOR I=7 TO 1 STEP -1
470 ST(I)=ST(I-1)
480 NEXT I
490 ST(0)=VAL(A$)
500 GOTO 160
510 '
520 'move stack down
530 '
540 FOR I=0 TO 7
550 ST(I)=ST(I+1)
560 NEXT I
570 GOTO 160
590 END
```

How the Program Works

The stack is stored in the array ST. Lines 140 to 220 print out the contents of the stack and the other screen contents and then wait for input. The LINE INPUT statement is used to get the input. This way the program can deal with either a number or a string: there is no way to tell in advance which the user will type.

The input string A$ is then analyzed to see if it's an arithmetic operator, like "+", or "*", or a function operator such as "sqr" or "exp". Arithmetic operators generally involve operating on the two lower items in the stack; this is handled in lines 240-300. Functions usually involve only the lowest item on the stack; these are handled in lines 320 to 420.

When an arithmetic expression combines two numbers, the contents of the stack must all be moved down. This is accomplished in lines 520 to 570, using a simple FOR...NEXT loop.

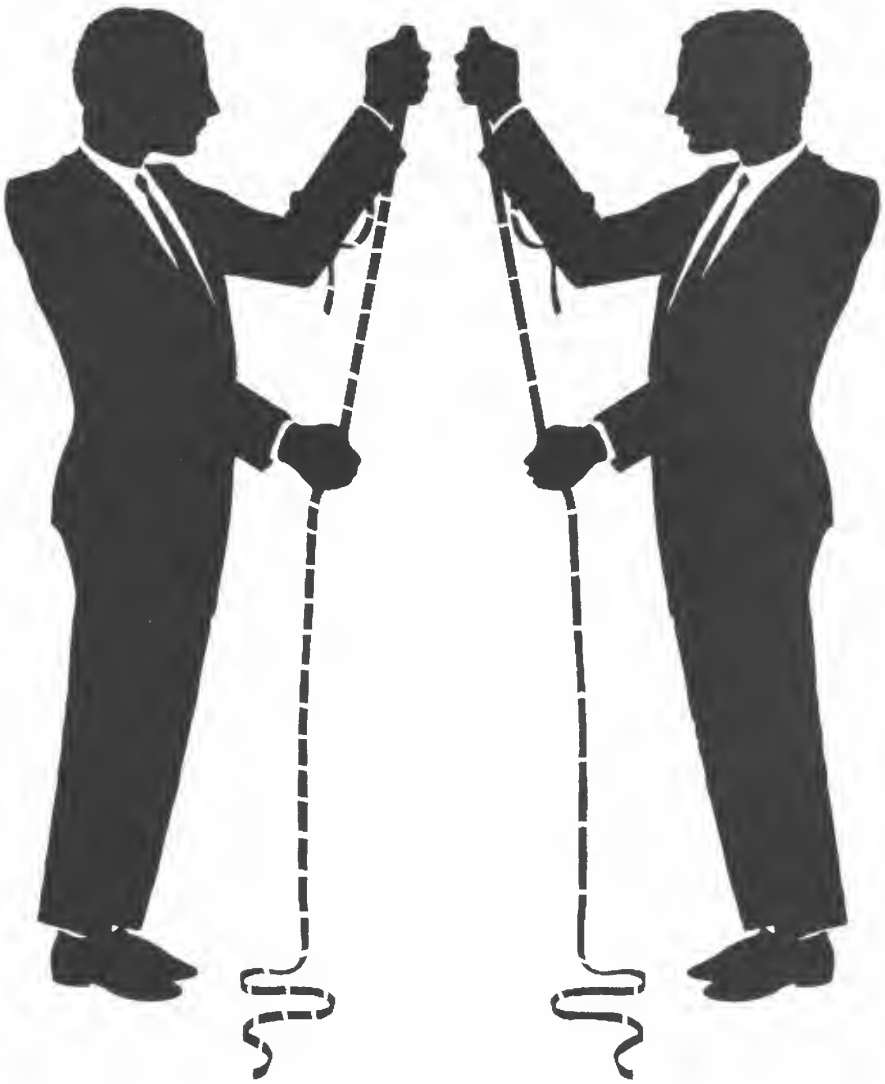
If the input is neither an arithmetic operator nor a function, the program assumes it must be a number. The stack is then moved up to make room for the number (lines 440 to 480), and the new number is added to the bottom of the stack. The number is converted from a string to a numerical value in line 490.

Adding New Functions

Adding new functions to the program is quite easy; simply follow the format of the existing program lines. If an arithmetic operator combines two numbers from the stack into a single value, then the other items on the stack must all be moved down one place, by jumping to the routine at 520. On the other hand, a function with a single input need only write the result into the bottom location on the stack by executing ST(0) = XXX(ST(0)), where XXX is the function.

9

Easy Conversions



METRIC

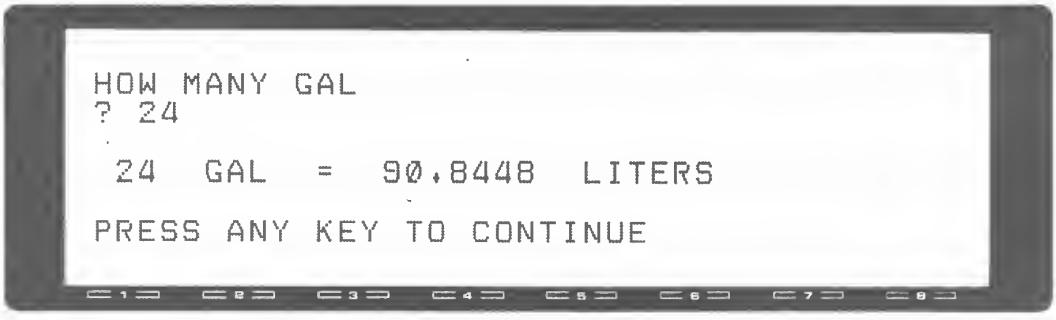
Mr. Metric

Metric ↔ English Converter

There are people who consider the metric system the work of the devil. On the other hand, it has its fanatical supporters, including Europeans and scientists. Whatever you think about it, the fact is that it's slowly infiltrating our red-blooded, apple pie, flag-waving American system of feet and inches, which is, of course, really the English system of feet and inches. Wine and gasoline are now sold by the liter, car engines are measured in cubic centimeters, track stars run the 100 meter dash, camera lenses are sold by the millimeter, and the weatherperson tells us it's going to be a nice warm day tomorrow at 20 degrees — centigrade.

How can an ordinary person keep track of all these ways of measuring things? Who can remember how many centimeters are in an inch, or how fast 80 kilometers per hour is in good old miles per hour? Mr. Metric contains sixteen of the most commonly used conversions, arranged in a simple menu-driven format so that it's easy to convert from a value in the English system to a value in metric (or vice versa) with just a few keystrokes.

When you start the program it asks whether you want to convert English to metric, metric to English, or one temperature to another. You enter either "E", "M", or "T". (That is, you type the letter and press ENTER.) Say you want to convert five and one-fourth inches to centimeters. This is English to



metric, so you type "E". The program will then display a group of possible English to metric conversions: gallons to liters, pounds to kilos, and so on. You enter the number 8, which is inches to centimeters. The program then asks you "How many In?" You enter the number 5.25, and the program immediately tells you this is 13.335 centimeters. The program will then pause until you tap any key, which will return you to the beginning of the program for the next conversion.

You can do any other conversion in a similar way.

If you want to perform some conversions which aren't in this program, you can add them to the existing program, or you can change some of the conversions already in the program to different ones. This is handy if, for example, you are in a business where conversion from furlongs to lightyears is a common problem.

Program Listing

```
10 '
20 ' METRIC
30 '
40 '
50 DIM A$(16),B$(16),C$(16)
60 FOR X=1 TO 16:READ C$(X):NEXT
70 FOR X=1 TO 16:READ A$(X),B$(X):NEXT
80 C=0:CLS:PRINT@12,"METRIC CONVERTER":PRINT:PRINT
90 PRINT"ENTER '1' FOR ENGLISH TO METRIC"
100 PRINT"      '2' FOR METRIC TO ENGLISH"
110 PRINT"      '3' FOR TEMPERATURE"
120 INPUT X$:IF X$= "2" THEN 130
    ELSE IF X$="1" THEN 210  ELSE IF X$="3" THEN 260  ELSE 80
130 CLS:P=40:FOR X=1 TO 7
140 PRINT@P,X;") ";C$(X);:P=P+20:NEXT
150 '
160 'Get Selection Number
170 '
180 PRINT@280,"ENTER THE NUMBER OF YOUR CHOICE";:INPUT C
190 C=INT(C):IF C<1 OR  C>7 THEN 130
200 GOTO 340
210 CLS:P=40:FOR X=8 TO 14
220 PRINT@P,X;") ";C$(X);:P=P+20:NEXT
230 PRINT@280,"ENTER THE NUMBER OF YOUR CHOICE";:INPUT C
240 C=INT(C):IF C<8 OR  C>14 THEN 210
250 GOTO 340
260 CLS:P=40:FOR X=15 TO 16
270 PRINT@P,X;") ";C$(X);:P=P+40:NEXT
280 PRINT@280,"ENTER THE NUMBER OF YOUR CHOICE";:INPUT C
290 C=INT(C):IF C<15 OR C>16 THEN 260
300 GOTO 340
310 '
320 'Get value, Gosub to formula
330 '
340 CLS:PRINT@120,"HOW MANY ";A$(C):INPUT B
350 ON C GOSUB 420,430,440,450,460,470,480,490,500,510,
    520,530,540,550,560,570
360 PRINT:PRINT B;" ";A$(C);" ";" = ";D;" ";B$(C)
370 PRINT:PRINT"PRESS ANY KEY TO CONTINUE"
380 X$=INKEY$:IF X$="" THEN 380  ELSE 80
390 '
400 'Formulas
410 '
```

```
420 D=B*.3937:RETURN
430 D=B*1.0936:RETURN
440 D=B*.061025:RETURN
450 D=B*.621377:RETURN
460 D=B*.26418:RETURN
470 D=B*.03527:RETURN
480 D=B*2.2046226:RETURN
490 D=B*2.54:RETURN
500 D=B*.91441:RETURN
510 D=B*1.6386:RETURN
520 D=B*1.6093:RETURN
530 D=B*3.7852:RETURN
540 D=B*28.3527:RETURN
550 D=B*.4535923:RETURN
560 D=(B-32)*(5/9):RETURN
570 D=B*(9/5)+32:RETURN
580 '
590 'Text to be Printed
600 '
610 DATA CENT. TO INCHES,METERS TO YARDS,
    CU CM. TO CU.IN.,KM. TO MILES
620 DATA LITERS TO GALS,GRAMS TO OUNCES,KILOGRAMS TO POUNDS
630 DATA INCHES TO CM'S,YARDS TO METERS,CU. IN. TO CU. CM.
640 DATA MILES TO KM'S,GALS TO LITERS,OZ'S TO GRAMS,
    POUNDS TO KILOS,DEGREES F TO C,DEGREES C.TO F.
650 DATA CM,IN,METERS,YDS,CC,CI,KM,MI,L,GAL,GRAMS,OZ,
    KG,LBS,IN,CM,YDS,METERS,CI,CC,MI
660 DATA KM,GAL,LITERS,OZ,GRAMS,LBS,KG,DEG F.,DEG C.,DEG C.,DEG F.
```

How the Program Works

In a program with so many conversion factors it would be easy to make a long program that would be time-consuming to type in. In this program DATA statements are used to efficiently store all of the text messages necessary for the various conversions. This way a given message can be referenced by a number, rather than by a string variable. This shortens the program considerably. Also, an ON GOSUB statement is used as a compact way to select the different conversion factors.

The first part of the program, from lines 10 to 70, initializes various variables, reads the strings from the DATA statements, and puts them into arrays where they will be useful for the program. The array C$ holds the definitions of each conversion, which are printed out in the menu along with their numbers, as for example "8) INCH'S TO CM'S". Arrays A$ and B$ hold the units themselves, such as "CM" and "METERS", with the units arranged in pairs: A$ holds the unit to be converted *from,* and B$ holds the unit to be converted *to.* Lines 60 to 70 read the data into these arrays.

Lines 90 to 120 are responsible for choosing which of the three categories, English to metric, metric to English, or temperature, will be used. This is done with multiple ELSE statements in line 120, which direct control to one of three sections of code.

Lines 160 to 300 consist of three similar sections, only one of which is executed on any given pass through the program. Each of these sections performs a similar function — asking the reader for the type of conversion he wants (inches to cm's, for example), and recording his answer as the value of the variable C. C is an important variable in this program; watch for its reappearance.

The heart of the program lies in lines 320 to 380, where the actual conversion is done. The program asks for the value of the input, and then depending on the variable C in the GOSUB expression in line 350, executes a short subroutine (one of the lines from 420 to 570) to actually perform the conversion. Now, watch how cleverly the program prints out the result. Suppose we are converting 6 inches to centimeters. Line 360 is a PRINT statement which prints out several items, all on one line. First, the variable B, which was the value typed in to be converted, or 6. Next A$(C), which is the units to be converted from, or "IN". Then an equals sign, and after

that, the variable D, which is the value of the answer, as found in one of the subroutines in lines 420 to 570. Finally, there's B$(C), the units to be converted to, in this case "CM".

Once the conversion is complete, the program waits for you to type any key, in line 370, and then goes back for a new conversion at line 80.

HEXDEC

Hexidec

Hex ↔ Decimal Converter

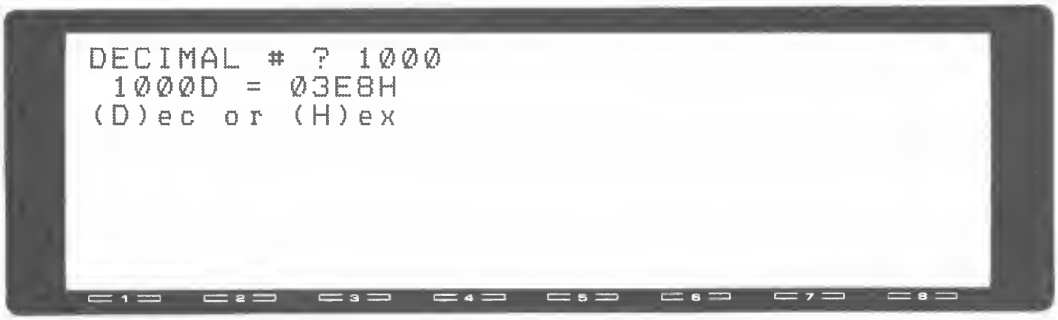
Do you want to amaze your friends with your knowledge of arcane computer lore? Are you interested in learning how computers *really* work? Do you want to broaden your perspective on the way numbers are used? Have you ever wondered how humans would do arithmetic if they had been born with sixteen fingers instead of ten?

If the answer to any of these questions is "yes", or if you work with computers a lot, especially if you speak assembly language or want to get involved in the deeper levels of the computer's operation, then you'll want to know about the hexadecimal numbering system, and you'll need a way to convert between hexadecimal and decimal numbers.

Why Hexadecimal?

Why does anyone want to worry about hexadecimal numbers? Humans have gotten along for generations with regular decimal numbers — why change now? The answer lies in how computers store numbers.

A computer thinks of numbers in terms of *bits*. A bit is like a little switch: it can be either on or off. Often this is written "0" for off and "1" for on. (Of course in a computer the switch is really a transistor.) You can arrange two



bits in four possible ways: 00, 01, 10 and 11. You can think of these four arrangements as standing for the decimal numbers from 0 to 3. Similarly, you can arrange three bits in eight possible ways, from 000 to 111, so you can count 0 to 7 decimal. And you can arrange four bits in sixteen possible ways, so you can count 0 to 15 decimal. Thus 0000 = 0, 0001 = 1, 0010 = 2, and so on up to 1111 = 15.

Now, the thing to notice here is that neither three bits nor four bits permits you to count exactly from 0 to 9 decimal (the ten-based system). Three bits is too few, and four bits is too many. Thus computer people had to make a choice: either a number system based on eight (three bits) or one based on sixteen (four bits). For a variety of reasons *hexadecimal*, the sixteen-based system, has emerged as the standard in the computer industry (although in years past many computers used the eight-based or *octal* system).

The next question is how can you write the numbers, with sixteen of them instead of ten? There will be six digits that can't be represented by the regular decimal digits. Someone came up with the idea of using the letters from A to F to represent these missing digits. Thus, "A" hexadecimal is 10 decimal, "B" hexadecimal is 11 decimal, up to "F" hexadecimal, which is 15 decimal. Table 9-1 shows the relationship between decimal, hexadecimal, and binary bit-patterns for the first sixteen numbers.

For convenience hexadecimal numbers are often represented with an "h" following them, while decimal numbers are followed by "d". Thus, we can say Ah = 10d (or sometimes AH = 10D). Also, "hexadecimal" is often abbreviated as simply "hex".

One of the major disadvantages of hexadecimal is the difficulty of converting back and forth between hex and decimal. For example, what's 1,475 decimal, expressed in hex? What's F3D2 hex, expressed in decimal?

The HEXDEC program gives you an easy way to do these conversions. Also, if you aren't familiar with hexadecimal, the HEXDEC program provides an easy way to find out what it's all about.

Operating the Program

When you run the program you will be presented with the choice of whether you want to go from hex to decimal, or decimal to hex. The screen displays:

```
(D)ec or (H)ex
```

At this point, you type "D" (or "d") if you want to go from decimal to hex, or "H" (or "h") if you want to go from hex to decimal. The program will only accept these two options and will not continue until one is entered correctly.

You will next be prompted to input the number in the base which you selected. The "D" option will only accept the digits from 0 to 9, while the "H" option will accept these digits plus the letters "A" to "F" (or "a" to "f"). Negative numbers will also be accepted.

As soon as a number is entered the answer will be displayed, and you will be prompted for another entry.

Decimal	Hexadecimal	Bit-Pattern
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Table 9-1. Relationships between decimal, hexadecimal and bit-patterns

Program Listing

```
100 '  HEXDEC
110 '
120 CLS
130 M$="":PRINT"(D)ec or (H)ex "
140 A$=INKEY$:IF A$="" THEN 140
150 IF A$="H" OR A$="h" THEN 200
160 IF A$="D" OR A$="d" THEN 330 ELSE 140
170 '
180 'hex to decimal
190 '
200 CLS:D=0:C=0:INPUT"HEX # ";H$
210 IF LEFT$(H$,1)="-" THEN H$=RIGHT$(H$,(LEN(H$)-1)):M$="-"
220 FOR X=LEN(H$) TO 1 STEP -1
230 A$=MID$(H$,X,1):GOSUB 270
240 M=16^C:D=D+(A*M):C=C+1:NEXT X
250 P$=M$+H$+"H= "+M$+STR$(D)+"D":PRINT P$: GOTO 130
260 'decode single hex digit A$
270 A=ASC(A$)-55
280 IF A>15 THEN A=A-32
290 IF A<10 THEN A=VAL(A$):RETURN ELSE RETURN
300 '
310 'decimal to hex
320 '
330 CLS:H$="":INPUT"DECIMAL # ";DE
340 DE$=STR$(DE)
350 IF LEFT$(DE$,1)="-" THEN DE$=RIGHT$(DE$,(LEN(DE$)-1)):
    M$="-":DE=ABS(DE)
360 D0=DE/16:D1=INT(D0):D2=(D0-D1)*16
370 IF D2>9 THEN D2=D2+55:A$=CHR$(D2):GOTO 390
380 A$=STR$(D2):IF LEFT$(A$,1)=" " THEN A$=RIGHT$(A$,(LEN(A$)-1))
390 H$=A$+H$
400 IF SGN(D0)=0 THEN P$=M$+DE$+"D = "+M$+H$+"H":PRINT P$:GOTO 130
410 DE=D1:GOTO 360
```

How the Program Works

The methods used to do the conversions in this program are fairly complex. To understand how the program works, it's helpful to select input values for the numbers and then follow the program through by hand, writing down the intermediate values of the variables. This should clarify the processes involved.

The first part of the program, from line 100 to line 160, initializes various variables, and then finds out from the user which conversion is desired: from hex to decimal, or from decimal to hex. An INKEY$ statement is used to read the letters "H" or "D"; this way the user need not type (ENTER) following the letter.

The balance of the program is divided into two separate parts, depending on which option is selected. Lines 180 to 290 perform the hex to decimal conversion, while lines 310 to 410 take care of the decimal to hex.

Hexadecimal to Decimal

There are two major tasks involed in the hex to decimal conversion. First the string which contains the individual digits must be taken apart so that each digit can be evaluated separately. This is handled in line 230, with a MID$ statement, which assigns the individual digits to variable A$.

Second, the particular digit in A$ must be turned into a number. This number is evaluated according to the digit's position in the hex number: a digit in the rightmost place is equal to its face value; a digit in the second column to the left is worth 16 times its face value (since it's a hexadecimal number); the third column to the right is worth 16 times 16 or 256 times its face value, and so on. The subroutine in lines 260 to 290 takes care of evaluating the individual digit. Line 240 calculates the value of the digit, using its position in the number, and adds it to the decimal number D. D is then printed out in line 250.

Decimal to Hexadecimal

The following method is used to convert from decimal to hex. First, a number (initially assigned to the variable DE) is divided by 16 (in line 360) to find its rightmost digit. This result is made into an integer (the remainder being thrown away), which is assigned to variable D1 and subtracted from the original number. This is the hex digit assigned to variable D2, which is

then converted to ASCII, either by adding 55 to it, if it is a letter from A to F (line 370) or by taking its value directly with STR$, if it is a digit (line 380). In either case, the resulting decimal digit, the string value A$, is then added to the variable which will be the final hex number, H$.

This entire process is now repeated for the next digit on the left. However, each time the process is repeated the initial number DE is made equal to the integer part of the preceding value of DE when it is divided by 16. That is, DE is set equal to D1 in line 410.

When all the digits have been added to H$, the result is printed out in line 400.

10

Dates, Times, and Schedules

DATE

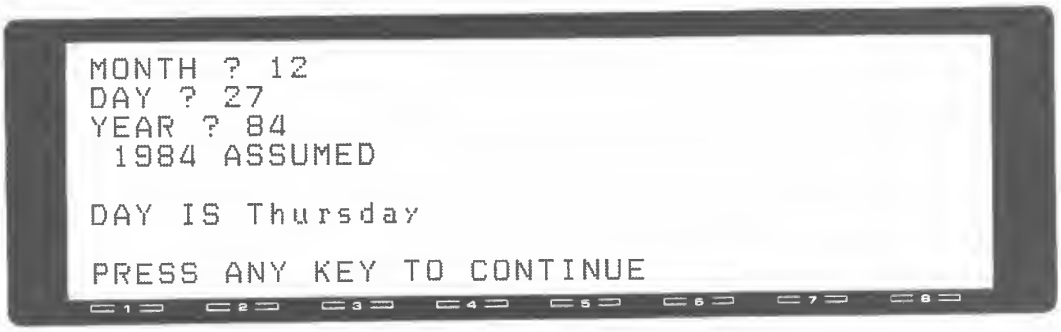
Perpetual Calendar

Day of Week for Any Date

Ever wonder on which day of the week your birthday will fall in the year 2000? Or what day of the week the Declaration of Independence was signed? Finding calendars for these years might be a difficult proposition! But take heart, for with this program your Model 100 can transform itself into an amazing "Perpetual Calendar".

To use the Perpetual Calendar program, enter the "MONTH", "DAY", and "YEAR", at the prompts. If you enter only two digits for the "YEAR", the program assumes you mean this century and adds 1900 to the number you enter. The day of the week, which corresponds to the date you entered, will be displayed. You may continue by pressing any key, or end the program by pressing (SHIFT) (BREAK).

Any year before 1753 will be rejected and you'll be asked to enter another "YEAR". This isn't an error in the program but was inserted deliberately. Why? Prior to 1753 consistent records of the extra ¼ day accumulated each year were not kept. (It's these fractions of a day that now give us leap year every four years.) These unrecorded six hours per year created many difficulties for people, notably astronomers, trying to keep track of the precise amount of time the earth took to orbit the sun. To correct this situation, around 1753 a change was made from the "old" Julian calendar



(based on the ancient Roman system) to our "modern" Gregorian calendar (named for the pope at that time). Dates before this change were based on a variety of systems, so a particular day of the week might have any number of dates ascribed to it! Therefore, the Perpetual Calendar program can only compute the day of the week accurately for dates after 1/1/1753. Also, the program only works up to 8/17/9989, so if you're planning something after that, you'd better fill in the day of the week later.

Program Listing

```
10 '
20 '  DATE
30 '
40 'Start Screen & Initialization of Date Variables
50 '
60 DEFINT D,M,Y
70 FOR X=1 TO 7:READ DA$(X):NEXT
80 CLS:PRINT@11,"PERPETUAL CALENDAR"
90 '
100 'Date Input
110 '
120 PRINT:INPUT"MONTH ";M:IF M<1 OR M>12 GOTO 80
130 INPUT"DAY ";D:IF D<1 OR D>31 GOTO 130
140 INPUT"YEAR ";Y:IF Y<100 THEN Y=1900+Y:PRINT Y;"ASSUMED"
150 '
160 'Julian Calendar Check
170 '
180 IF Y<1753 THEN PRINT"MUST BE AFTER 1752":GOTO 140
190 '
200 'Determine Century & Year/Leap Year Check
210 '
220 A=INT(Y/100):B=Y-100*A:F=0
230 IF M>2 THEN 300  ELSE F=2:IF B=0 THEN 250
240 C=B-4*INT(B/4):IF C<>0 THEN 300  ELSE F=1:GOTO 300
250 C=A-4*INT(A/4)
260 IF C=0 THEN F=1
270 '
280 'Compute Day Factor & Establish Day of Week
290 '
300 D=INT(365.25*B)+INT(30.56*M)+F+D
310 E=3+D-7*INT((D+2)/7)
320 '
330 'Display Results & Continue Loop
340 '
350 PRINT:PRINT"DAY IS ";DA$(E)
360 PRINT:PRINT"PRESS ANY KEY TO CONTINUE"
370 X$=INKEY$:IF X$="" THEN 370  ELSE 80
380 '
390 'Data Statement for Days of the Week
400 '
410 DATA Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday
```

How the Program Works

Line 60 defines the variables D, M, and Y as integers. The FOR...TO and READ commands in line 70 refer to the DATA statement in 410 where the days of the week are listed. Lines 120 through 140 request data for the month, day, and year variables. The "Julian Calendar" check occurs in 180.

The portion of the program that determines if the year entered is a leap year is intriguing. First, the four digit "YEAR" is broken down into "century" and "decade" digits, the variables A and B respectively in line 220. For 1984 A would equal 19 and B, 84. By utilizing the following formulas, the "century" and "decade" digits can establish whether a given year is a leap year. The given year is a leap year if B is *not* equal to zero but *is* divisible by four, or if B *is* equal to zero and A is divisible by four. 1984 is a leap year because B (84) is evenly divisible by four. The year 2000 is a leap year because A (20) is divisible by four. Once the leap year check has been completed, the F (flag) variable must be adjusted. F equals two if the month entered is January or February and the "YEAR" is not a leap year. F equals one if it is a leap year. The F variable enables the program to account for the extra day in February during a leap year.

The day of the week for the date entered is derived from the equations in lines 300 and 310. As mentioned earlier, the Perpetual Calendar Program is only accurate to 8/17/9989. Any date after this will cause an OV (overflow) error in line 300 because the numbers calculated in the formula exceed the digit limit on your Model 100.

Line 350 displays the result of the computations as a day of the week. The INKEY routine in line 370 allows you to rerun the program by pressing any key. The READ-DATA technique is used to move the days of the week in line 410 into a string array. This method is generally preferable to multiple statements such as A$(1) = "Monday", A$(2) = "Tuesday", and so on, especially in applications where a large number of variables must be defined.

BETWEEN

Days Between

Days Between Two Dates

How many days until your vacation? How many days have passed since you were born? Have you ever been concerned about long term capital gains and tried to figure out how many days have elapsed since you made a particular investment? Or how many days have passed since you signed a particular contract?

With your Model 100 and this Days Between program you won't need to leaf through your calendar counting up the days anymore. You simply run this program, type in the appropriate dates, and let the Model 100 do all the work.

When you start the program, you will be prompted to enter the "FROM" (the starting date), and then "TO" (the ending date). The Model 100 calculates the number of days between the dates, displays the result, returns to the FROM prompt, and waits for your next entries. If you enter an ending date that falls before the starting date, you will receive a negative number as an answer.

As in the Perpetual Calendar program described earlier, Days Between limits you to dates after 1753, checks to see if you have entered valid dates, and assumes that a two digit "YEAR" entry is in the twentieth century.

Program Listing

```
10 '
20 '  BETWEN
30 '
40 ' Start Screen & Definition of Date Variables
50 '
60 DEFINT M,D,Y
70 CLS
80 PRINT"FROM:":GOSUB 260
90 GOSUB 210:X1=X
100 PRINT "TO":GOSUB 260
110 GOSUB 210:X2=X
120 '
130 'Difference Between Dates Calculated & Displayed
140 '
150 B=X2-X1
160 PRINT:PRINT"DAYS BETWEEN DATES = ";B
170 GOTO 80
180 '
190 'Elapsed Days from Year 0 Calculation
200 '
210 X=Y*365+INT((Y-1)/4)+(M-1)*28+
    VAL(MID$("000303060811131619212426",(M-1)*2+1,2))-
    ((M>2)AND((Y AND NOT-4))=0)+D
220 RETURN
230 '
240 'Date Entry Sub-Routine & Julian Calendar Check
250 '
260 INPUT "MONTH ";M:IF M<1 OR M>12 GOTO 260
270 INPUT "DAY   ";D:IF D<1 OR D>31  GOTO 270
280 INPUT "YEAR  ";Y:IF Y<100 THEN Y=Y+1900:PRINT Y;"ASSUMED"
290 IF Y<1753 THEN PRINT"MUST BE AFTER 1752 ":GOTO 280
300 RETURN
```

How the Program Works

This program operates by taking any two valid dates and converting them into numbers relative to a hypothetical starting point. The central portion of this calculation takes place in line 210 where the total number of elapsed days are counted from January 1st, year zero, as if there had been a constant calendar since that hypothetical date. Of course this was not the case; dates before 1753 were not accurate, but this does not affect calculations of the difference between dates that occur after that year.

The equation in line 210 is computed in the following manner. First, the sum of days from the hypothetical zero starting date to the entered year is totaled; then the number of days in the months of the last year of the entered date are calculated assuming there are twenty-eight days in each month. This "monthly" figure is then adjusted for the extra days over twenty-eight, using the string in quotes. An adjustment is made for leap years (those divisible by four), and finally the day of the month is added. Whew! Now that you know how this one line works, you can use it as a keystone in any program that needs to determine the number of days between two dates.

Line 150 figures the number of days between the two dates entered by subtracting the results from line 210 for the two date sets. Rather than defining two sets of variables as M1 and M2, D1 and D2, and so on, for the "MONTH", "DAY", and "YEAR" entries, a subroutine is used in lines 260 through 300. The GOSUB...RETURN combination saves you time when entering code and limits confusion when dealing with more than one set of like variables.

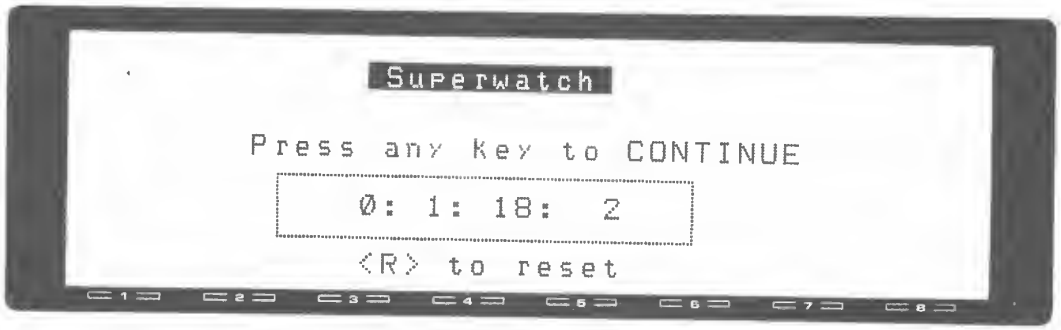
TIME

Superwatch

Sophisticated Stopwatch

On your mark, get set, GO! As the runners line up for the World finals of the 800 meter event, you call up your Model 100 Superwatch program. The favorite, Lightning Jones, gets set in the blocks, the gun sounds, and you press a key on your Model 100. The timer starts, the runners are out in good order. Jones is third at the 200 meter mark. You hit a key, the Model 100 reads 00:21:4. At the first lap Jones is still third with 00:43:6 for 400 meters. It will take a tremendous second lap for the race to come near the world record time. Jones begins his kick at the 600 meter point, passes the first two runners, and crosses the finish line with 01:14:2 reading on the Superwatch. An outstanding time!

This is just one exciting way you can use the Superwatch program as a lap timer. Time your favorite racing events, or use the timer in the kitchen to prepare meals with precision. When you start the "clock", it will continue to run accurately for up to twenty-three hours or until you reset the program. It reads out in hours, minutes, seconds, and tenths of a second, and is accurate to one-fifth of a second. You can pause to view a lap time at any moment by hitting a key; to continue with the lap simply press any key



again. If you'd like to begin timing a new event, restart the clock by pressing "R". The time will reset to zero, and you'll be ready for another "on your mark, get set, and go!"

Program Listing

```
10 '  TIME
30 '
40 'Start Screen
50 '
60 CLS:ET=0:CALL 17001:PRINT@53," Superwatch ";:CALL 17006
70 PRINT@128,"Press any key to START   "
80 '
90 ' Initializes Watch & Displays the Elapsed Time
100 '
110 GOSUB 400:GOSUB 290:T1=V:TH=V
120 LINE (56,36)-(166,52),1,B
130 PRINT@145,"STOP     ":PRINT@291,SPACE$(15);
140 TS=1
150 X$=INKEY$:IF X$<>"" THEN 200
160 GOSUB 290:ET=V-T1:GOSUB 320
170 IF V>TH THEN TS=0:TH=V
180 PRINT@213,SPACE$(10);:PRINT@212,ET$;:PRINT@ 223, TS;:
    TS=TS+2:IF TS=10 THEN TS=0
190 GOTO 150
200 GOSUB 290:TS=1:T2=V:ET=T2-T1:GOSUB 320
210 '
220 'Restart/Continue Loop
230 '
240 PRINT@212,ET$;:PRINT@145,"CONTINUE";:
    PRINT@293,"<R> to reset";
250 GOSUB 400:IF X$="R" OR X$="r" THEN 60 ELSE GOTO 130
260 '
270 'Time Conversions
280 '
290 T$=TIME$:H=VAL(LEFT$(T$,2)):M=VAL(MID$(T$,4,2)):
    S=VAL(RIGHT$(T$,2))
300 V=(H*3600)+(M*60)+S
310 RETURN
320 H1=INT(ET/3600):H$=STR$(H1)
330 S1=ET MOD 60:S$=STR$(S1)
340 M1=INT((ET-(H1*3600)-S1)/60):M$=STR$(M1)
350 ET$=H$+":"+M$+":"+S$+":"
360 RETURN
370 '
380 'Stop Control for Watch
390 '
400 X$=INKEY$:IF X$="" THEN 400 ELSE RETURN
```

How the Program Works

The root of this program depends on the Model 100's built-in clock and TIME$ function. This function retrieves the present time as a character string. In order to use the string accessed by the TIME$ function, it must be converted to a numeric value. The conversion begins in line 290. The string is broken into values corresponding to hours, minutes, and seconds. These values are then converted into seconds and combined to give a value representing the total seconds. The subroutine in lines 320 through 360 computes how many hours, minutes, seconds are in the "total second" value, reassembles a string so that the current elapsed time may be displayed, and loops back to the TIME$ function and string conversion in 290.

Lines 60 to 200 set up the display screens and initialize the variables for the elapsed time and one-tenth seconds. The key variables in this section are T1 (initial time), TS (tenth seconds), and ET (elapsed time). Values for these variables are determined in the Time Conversions section, lines 290-360.

The Model 100's built-in clock will *not* display fractions of seconds! How can this program display a one-tenth second? A "false" one-tenth second can be derived by taking advantage of the fact that the time conversion subroutines loop approximately five times each second. Line 180 counts the number of loops that occur, displays the values, zeros the one-tenth second value when the total equals ten, and then repeats.

A note of caution! In order to save a large amount of coding, no effort was made to correct the timing error that will occur if a "lap" passes midnight. The built-in clock reads from 23:59:59 to 00:00:00. The twenty-four hour differential throws the elapsed time computations off. If you need the ability to time through midnight, you can modify the program to test for zeros in the variables after GOSUB 290 in line 160. If zeros are detected, reduce the initial time, T1, by 86,400 (the number of seconds in twenty-four hours).

11

Managing Text Files

EZSORT

Easy Sort

Sort Items in a Text File

Sorting raw data into ordered lists is the most often used application of computer technology today. This is true for computers ranging from large mainframes to small portable micros. Your Model 100 is no exception. Do you have a list of names you'd like to put in alphabetical order? Would you like to rank a list of your students by order of their final exam grades? With our Easy Sort Program you can sort any document (.DO) file. You'll find this easy-to-operate program especially useful for your ADRS.DO and SCHED.DO files.

When you run Easy Sort, you will be asked to input the name of the document file you wish to sort. Be sure to include the .DO extension or the program will not be able to locate the file; instead you will get a "NM" (bad file name) error. If you enter a valid filename, the entire file will be displayed one record at a time. After the last record has appeared on the screen there will be a pause, then a series of stars (*) will be displayed. Each star represents one loop of the sort routine. When the file sort has been completed, the file will be displayed in the new sorted order as it is being written back into memory under the original file name. Finally, the "DONE" message will come up on your screen and the program will end.

Note that this program is a simple string sort. It takes each record and

compares it to all the other records beginning with the first character. For this reason you should think carefully about the design of your data files. For instance, if you entered the records for a file in this order — First Name, Last Name, Street, etc. — and then ran Easy Sort on the file, Bill Jones would come before John Adams in the sorted file. The Last Name should have come first in order for Easy Sort to be really useful. Proper prior file planning will enable you to use Easy Sort in a variety of helpful ways.

If you need more sophistication in a sorting program, see the program Any Sort which follows this one.

Program Listing

```
10 '  EZSORT
30 '
40 'File/Memory Allocation and Definition
50 '
60 MAXFILES=1:CLEAR 2000:C=0
70 M=200  'Set this to max size of file
80 '
90 'Start Screen & Filename Input
100 '
110 CLS:DIM S$(M)
120 INPUT",DO File to sort - ";S$
130 '
140 'Read Routine
150 '
160 OPEN S$ FOR INPUT AS 1
170 FOR L=1 TO M
180 LINEINPUT #1,S$(L)
190 PRINT S$(L)
200 C=C+1
210 IF EOF(1) THEN 250 ELSE NEXT L
220 '
230 'Shell-Metzner Sort
240 '
250 X=C
260 X=INT(X/2):IF X=0 THEN 370
270 Y=1:Z=C-X
280 V=Y
290 W=V+X:IF S$(V)<=S$(W) THEN 320
300 H$=S$(V):S$(V)=S$(W):S$(W)=H$
310 V=V-X:IF V<1 THEN 320  ELSE 290
320 Y=Y+1:IF Y>Z THEN 260
330 PRINT"*";:GOTO 280
340 '
350 'Close Input File, Open & Display Output File
360 '
370 CLOSE:OPEN S$ FOR OUTPUT AS 1
380 PRINT:FOR L=1 TO C+1
390 PRINT #1,S$(L):PRINT S$(L)
400 NEXT
410 PRINT "DONE"
```

How the Program Works

Lines 60 and 70 initialize variables, including the variable M, which must be set to *greater than the maximum number of items you want to sort.* In the listing it is set to 200; if you need to sort more items and have enough memory, change it accordingly. The constant M is used to define an array S$(M) in line 110.

The name of the file to be sorted is obtained from the user in line 120. Then the items from the file are read, displayed, and inserted into array S$ in lines 160 to 210.

This program uses a very efficient sorting process called the Shell-Metzner sort. This is carried out in lines 260 through 330. The Shell-Metzner sort is considerably faster than the more commonly used "bubble" sort. Even though it is speedier, it is just as short and easy to use as other sort routines.

The Shell-Metzner algorithm operates by dividing all the items to be sorted into a number of smaller sorts that progressively bring the entire file into order. This operation is hard to describe, but can be seen in Figure 11-1 where the data is continuously broken into smaller numbers of lists and greater numbers of sorted fields.

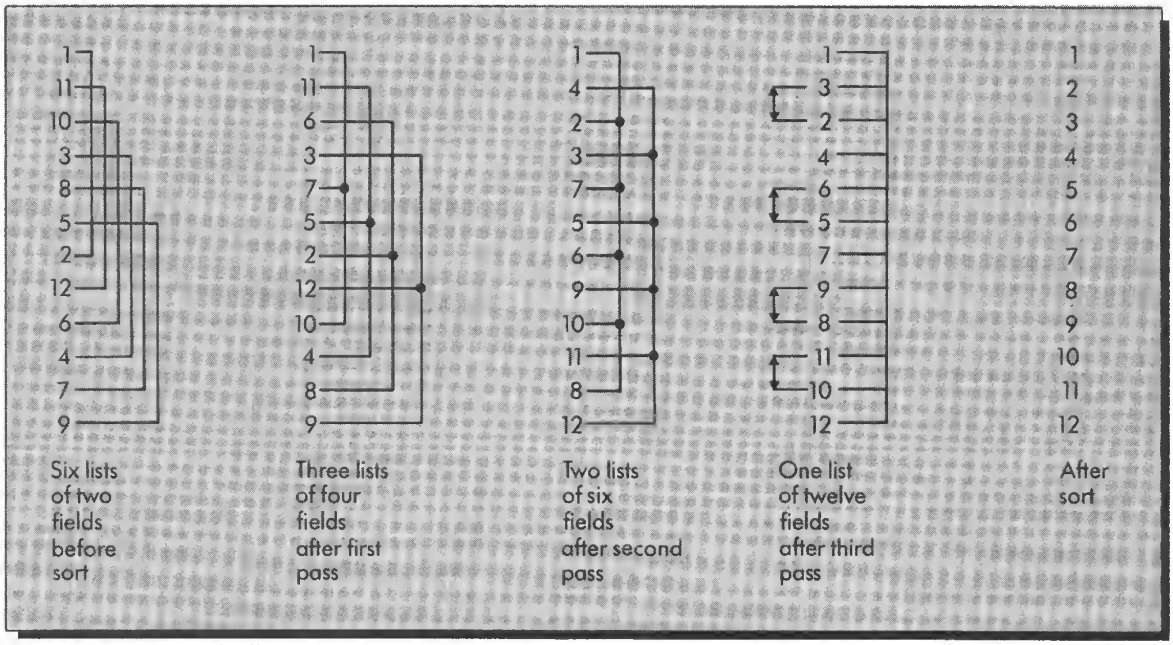


Figure 11-1. Shell Metzner sort

If you are working with large data files, you might have to increase the amount of memory that is cleared in line 60 and enter a larger value for the maximum number of records in line 70. You must have at least as much memory free as is occupied by the unsorted file, plus about five bytes per record for an operating buffer. Larger files will cause a considerable slowdown in the run time of the program. One way to speed up the program is to eliminate ":PRINT S$(L)" from line 390. You will no longer see the sorted file on the screen, but the program will run much faster.

Once the file has been sorted, the records are taken from the array S$ and written back out to the same file that was originally opened. This is done in lines 370 to 400.

The sort routine in this program can be removed from the program and modified for use with any string array you might want to sort. Simply change the named variable S$ in lines 250 to 300 to match the name of the array you want to sort. Then make the line number, referred to in the IF statement in line 260, coincide with the location that you wish to go to after the sort is completed. You'll find this sort routine useful in a wide variety of applications.

ANY

Any Sort

Sorts Items by Column Number

Here's another Model 100 program that will sort text files. The main difference between Any Sort and the Easy Sort program discussed earlier, is that Any Sort sorts a data file *by a column that you specify*. You can also sort by ascending or descending order. These features are especially useful when you want to sort the same file in several different ways. However, to use this feature your files must be organized so that similar items fall in the same column.

For instance, you may have an address file in your Model 100 which is organized like the following sample:

```
Petersen,Harry :722-9000: 251 E. 5th
Smith,Ted      :999-1212: 314 Main St.
Hansen,Mary    :444-2345: 7   Elm St.
Young,Bob      :321-5674: 235 So A Ave.
Anderson,Al    :893-8492: 846 Lancaster
----------------------------------------
12345678901234567890123456789012345678901
         1         2         3         4
```

Let's say you want to sort the file, first by last name, then by address, and, finally, by phone number, in descending order, and that you want to store

```
Input Filename? list.do
Sorted Filename? list2.do
A=Ascending or D=Descending order? a
First Column of Sort Field? 8
Length of Sort Field      ? 9
OK
```

each of the sorted files under a name that's different from the original. Any Sort can handle this easily.

When you run the Any Sort program, you will be prompted for these entries:

```
Input Filename?
Sorted Filename?
A=Ascending or D=Descending order?
First Column of Sort Field?
Length of Sort Field?
```

At the "Input Filename" prompt, enter the name of the file you want to sort, then type in the name you want for your sorted file. Tell the program if you want the file sorted in ascending or descending order by entering "A" or "D" respectively.

We have added a "measuring stick" below the sample file above to help you locate columns. To sort the sample file by last name, you'd enter "1" at the "First Column of Sort Field" prompt. To sort by phone number, you would specify "17" as the first column of the sort field. You'd enter "27" for a file sorted by address number, or "31" to sort by street name. The answer to the "Length of Sort Field" query determines the number of characters over which the sort will take place, and depends largely on how you have organized your text files.

For instance, if you want to sort the sample file by address number in ascending order, specify column "27" as the first column of the sort field with a length of "3". The resulting file would look like this:

```
Young,Bob       :321-5674: 235 So A Ave,
Petersen,Harry  :722-9000: 251 E, 5th
Smith,Ted       :999-1212: 314 Main St,
Hansen,Mary     :444-2345: 7   Elm St,
Anderson,Al     :893-8492: 846 Lancaster
```

However, if you specified the sort field length as "1", the sorted file would be:

```
Petersen,Harry  :722-9000: 251 E, 5th
Young,Bob       :321-5674: 235 So A Ave,
Smith,Ted       :999-1212: 314 Main St,
Hansen,Mary     :444-2345: 7   Elm St,
Anderson,Al     :893-8492: 846 Lancaster
```

The program would have no way of determining that there "should" be a difference between the Petersen and Young records. The "2's" in column "27" are the same, so Any Sort wouldn't reorder the corresponding records.

Perhaps you're asking yourself how

```
Hansen,Mary     :444-2345: 7   Elm St.
```

could be sorted properly. The answer is that it can't. At least not the way it appears now. However, if the "Hansen" record were changed to read

```
Hansen,Mary     :444-2345: 007 Elm St.
```

and you had specified the first column to sort as "27" and entered a sort field length of "3", then Any Sort would sort the file in this order:

```
Hansen,Mary     :444-2345: 007 Elm St.
Young,Bob       :321-5674: 235 So A Ave.
Petersen,Harry :722-9000: 251 E. 5th
Smith,Ted       :999-1212: 314 Main St.
Anderson,Al     :893-8492: 846 Lancaster
```

This illustrates the importance of planning your files very carefully. No amount of sorting, regardless of sophistication, will help a file that begins as a hodge-podge!

Please take note that Any Sort should *not* be used on text files that contain commas. The program reads commas as end-of-records and skips out of the proper column/record sequence. If you plan to use Any Sort on your files leave commas out.

When Any Sort has completed the ordered sort, it closes the files, returns to BASIC, and displays the flashing "Ok" prompt. If you have another file to sort or want to sort the same file again, just run the program again. Be sure you have enough memory to run the program, at least 4K depending on the size of the file you want to sort.

Program Listing

```
10 '
20 '  ANY
30 '
40 'File Definition & Input
50 '
60 CLS
70 MAXFILES = 2
80 DIM IN$(200),SD$(200)
90 INPUT "Input Filename";NF$
100 OPEN NF$ FOR INPUT AS 1
110 INPUT "Sorted Filename";SF$
120 '
130 'Order of Sort
140 '
150 INPUT "A=Ascending or D=Descending order";SQ$
160 IF SQ$="a" OR SQ$="A" THEN SQ$="A"
170 IF SQ$="d" OR SQ$="D" THEN SQ$="D"
180 IF SQ$<>"A" AND SQ$<>"D" THEN GOTO 150
190 '
200 'Column/Field Definition
210 '
220 INPUT "First Column of Sort Field";ST
230 INPUT "Length of Sort Field      ";LG
240 '
250 'Read Routine
260 '
270 IF EOF(1) THEN 310
280 CT=CT+1
290 INPUT #1,IN$(CT)
300 GOTO 270
310 CLOSE 1
320 '
330 'Column Sort
340 '
350 OPEN SF$ FOR OUTPUT AS 2
360 FOR I=1 TO CT
370 IF SQ$="A" THEN X$=STRING$(LG,255) ELSE X$=STRING$(LG,0)
380 FOR J=1 TO CT
390 IF SD$(J)="Y" THEN 440
400 IF SQ$="A" THEN IF MID$(IN$(J),ST,LG)>X$ THEN 440
410 IF SQ$="D" THEN IF MID$(IN$(J),ST,LG)<X$ THEN 440
420 K=J
430 X$=MID$(IN$(J),ST,LG)
440 NEXT J
```

```
450 PRINT #2,IN$(K)
460 SD$(K)="Y"
470 NEXT I
480 END
```

How the Program Works

Line 70 sets the maximum number of files to be opened at two: one input file and one output file. The "DIM" statement in 80 defines the number of records for each file. If you are sorting large data files you'll want to increase the number of records to an amount appropriate to the size of your file.

Input values at the various prompts are gathered in lines 100, 110, 150 through 180, 220, and 230. The read routine (lines 270 through 310) reads the input file into memory, establishes a counter as the file is read, checks for the end-of-file marker, and closes the file when the marker is found.

The sort operates by loading the input file into an array in line 370. If an ascending sort was specified, the lowest ASCII valued element in the column is found (line 400) and written to the output file. The search continues by finding the next highest value which is, in turn, written to the output file, and so on until the end of the file (determined by the input file counter in line 280 and the "FOR...TO" in 360). For descending output the highest valued ASCII element is found (line 410) and "printed" to the output file, then the next highest, and so on to 1, to yield a file sorted in descending order.

COUNT

Word Count

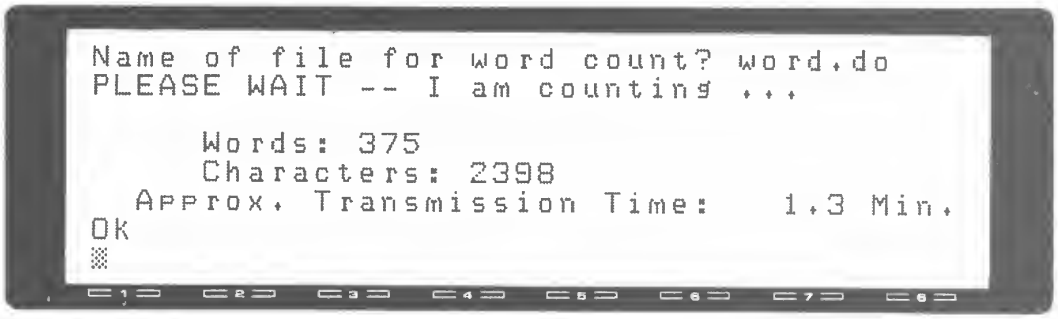
Counts Words in File

This handy little utility program counts the number of words and characters in any file saved with the .DO extension. Word Count also estimates the time it would take to transmit the file using the Model 100's 300 baud modem. For instance, applying Word Count to a typical .DO file shows this display:

```
   Words: 375
   Characters: 2398
Approx. Transmission Time:    1.3 Min.
```

You can get an idea of how large your files are, and the amount of time it will take to transfer them, before you actually transmit to another computer.

To use the Word Count program, simply respond to the "Name of file for Word Count" prompt by entering the filename of the file you want to know about. The message "PLEASE WAIT — I am counting..." now appears on the screen. *Be Patient!* The program takes some time to complete its run — approximately one minute for every 200 words in your file. After Word Count displays its results, you can rerun the program by entering "R" or end the program by pressing any other key.



Program Listing

```
10 '
20 '  COUNT
30 '
40 'Initial Screen, File Name Entry & Counter Set
50 '
60 CLS:B$=""
70 WD%=1
80 INPUT "Name of file for Word Count";NF$
90 PRINT "PLEASE WAIT -- I am counting ..."
100 '
110 'File Opened & Variables Defined
120 '
130 OPEN NF$ FOR INPUT AS 1
140 A$=INPUT$(1,1)
150 B$=B$+A$
160 CR%=CR%+1
170 '
180 'Character "Search" Routine
190 '
200 IF CR%>3 THEN B$=RIGHT$(B$,3)
210 IF EOF(1) THEN 280
220 IF A$=CHR$(10) THEN WD%=WD%+1
230 IF LEFT$(B$,1)<>CHR$(32) AND MID$(B$,2,1)=CHR$(32)
    AND RIGHT$(B$,1)<>CHR$(32)THEN WD%=WD%+1
240 GOTO 140
250 '
260 'Close File & Display Results
270 '
280 CLOSE 1
290 PRINT@125,"Words:";WD%
300 PRINT@165,"Characters:";CR%
310 CR=CR%
320 TI=CR/1800
330 PRINT "  Approx. Transmission Time:";
340 PRINT USING "####.#";TI;
350 PRINT " Min."
360 '
370 'Continue/End Loop
380 '
390 PRINT "Enter 'R' to rerun"
400 I$=INKEY$: IF I$="" THEN 400
410 IF I$="R" OR I$="r" THEN 60 ELSE END
```

How the Program Works

Lines 60 through 90 clear the screen, display the query for a filename, and the "PLEASE WAIT" message. The file that you enter is opened in line 130. Lines 70 and 160 initialize the counting variables WD$ for the word count and CR% for totaling the characters (symbols, numbers, letters, and spaces) in the file. The Word Count program determines the number of words by counting the spaces between the words in your file and adding one to a preset variable WD%. Line 230 represents the keystone in this space key reading operation. Line 160 simply adds 1 to CR% every time another character is read from the file.

The program loops (line 240) until EOF (end of file) is read (line 210), then prints the results of its counting in lines 290 and 300. Line 240 derives the approximate transmission time for the file by dividing the number of characters in the file by 1800 (the baud rate of the Model 100's built-in modem, multiplied by the number of seconds in a minute: 300*60).

WSTAR

Formatter

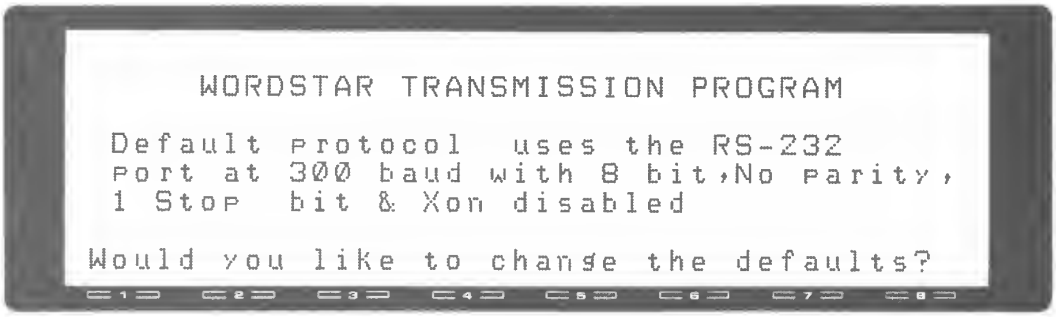
Converts Text Files for WordStar®

The airline information screen is flashing "DELAYED" next to Flight 126 out of Minneapolis. Not only are you stuck in the Minneapolis airport in January, but you also made the mistake of taking out your Model 100 in the middle of a jammed waiting area. Now everyone wants to know if they can have a look at it!

At last the excitement quiets down, and you settle in to type some notes about your East Coast trip. You enter a summary of the trip's main meetings, a letter to the New York Production Manager, and create a file for the sales figures you managed to dislodge from the Boston division's Marketing Director.

When you get back to the office you plan to do the final editing and embellishment on these files in WordStar, using your desktop microcomputer and this Formatter text transfer program. You glance at the airline's monitor and then at your Model 100's display clock; the flashing "DELAYED" is gone, and Flight 126 is boarding.

When you use the Formatter program the files you create on the Model 100's built-in text editor can be used on any microcomputer with WordStar, or by a variety of other word processing packages, depending upon the file format the software requires.



Why can't you simply use the Model 100's "Telcom" program to transfer these files? In addition to the fact that the Model 100 stores documents in a manner incompatible with WordStar, Telcom inserts hard carriage returns into the text at the end of each line. This conflicts with WordStar files, which use a special end-of-line marker after each line: a soft carriage return (8D hexadecimal). Paragraphs in WordStar are, on the other hand, denoted by the standard line feed and carriage return characters 0A and 0D hexadecimal. Because the Telcom program ends each line with an 0DH and no line feed, WordStar cannot be conveniently used to reformat a file transmitted in this manner.

In order to run your Formatter program, you must connect your Model 100 to another computer. There are two common ways to do this: a modem to modem connection, or direct connection with the RS-232 ports on both machines. We'll discuss both of these methods and their advantages and disadvantages.

Direct Connection

Perhaps the best means of utilizing the Formatter program is to "direct connect" the two machines via the RS-232 ports. You do not need a modem or communications software to do this (though if you have the software, you can easily use it to save the files coming in from the Model 100).

If your desktop microcomputer uses CP/M, you can use the PIP command (Peripheral Interchange Program) that is part of the CP/M operating system to create a disk file from the input device. This will then be buffered to memory until an end-of-file marker is detected. Use this command:

```
PIP FILE.EXT=RDR: B
```

where "FILE.EXT" is the name you choose for the file under CP/M. Buffering the incoming data is necessary because the RS-232 port on the Model 100 does not recognize any of the serial "handshaking" signals that regulate the start and stop of data transmissions in typical RS-232 serial communications.

Now, with the RS-232 ports connected by a serial cable, run the Formatter program. The program will ask whether you want to change the default settings. Answer the prompt in the affirmative by entering "Y" (for Yes). Check out the default settings for the communications protocol menus, and note the wide variety of protocol settings the transfer program makes available. (To run the program without changing any of the default settings,

answer "no" by entering "N", and then respond to the next prompt with the name of the file you want to transfer.)

The first protocol menu is for baud rate selection with a default setting of 300 baud (marked with an asterisk). If your other machine can receive data at a high baud rate, you may want to transmit at the highest rate possible. The higher the baud rate, the faster your files will be transferred. Choose the number appropriate to the rate at which you want to transfer. You'll then be asked for the "word length" (or bit length), "parity", "stop bits", and "line status" (or X-On/X-Off Protocol). Unless you know that you need to change any of these settings, you should enter the default settings. Finally, the program will display the communication settings that you've chosen, and ask if the selections are correct. If you're not sure, enter "N" for "no," and the program will bring you back to the baud rate display. If the settings are right, answer "yes" by pressing "Y" and enter the name of the file you wish to transfer at the next prompt.

The complication of using the "direct connect" method is that, depending on the type of computer you are transmitting to, it is possible that a standard RS-232 cable will not function correctly. You may need to use a "null modem" cable between the two machines. If so, consult your machine's manual for a description of the serial cable's required pin numbers; then, if necessary, reconfigure your cable (usually by taking it apart and reversing the lines to pins #2 and #3). You can also purchase the proper cable directly from an electronics store.

Modem to Modem

To run the transfer program with a modem to modem connection, your other machine will need a modem and a communications package capable of receiving text files. Your Model 100's built-in modem operates at 300 baud (you'll have to change the baud rate default setting to "0" for "Modem 300"), so your files will necessarily be transferred at this rate. If you have the equipment and software to follow this method, then start with both modems directly connected by the beige line from your Model 100's modem jack. Though you could try "long distance" transfer of files (using a telephone line rather than both computers actually being in the same room) the tricky logistics necessary for such a transfer may detract from its usefulness. Split-second timing will be required by a helper at the machine receiving the file, since the WordStar Transfer program does not "handshake" (recognize the other machine's communications protocol). As soon as you send the Model 100 file your helper will have to "capture" the incoming data. If

there is any delay a portion of the file will be lost. Still, you may find the idea of "interstate" file transfer so attractive that you'll want to try this method out. If so, set your Model 100 to the "direct connect" and "answer" modes by adjusting the switches on the left side. These settings will provide a carrier signal when your Model 100 enters the "terminal" mode. This signal should be recognized by the other modem. Once all your equipment is in proper order, you can run the WordStar Transfer program and transfer your files without having to change any of the program's default settings.

When you have written your file to disk using either method, you can load the file into WordStar and reformat it by executing the WordStar global reformat: CTRL Q, Q, CTRL B. This "new" file format enables you to edit your file using WordStar commands.

Program Listing

```
10 '
20 '  WSTAR
30 '
40 'Default Configuration Setting
50 '
60 MAXFILES=2:P$="COM:88N1D"
70 '
80 'Title & Defaults Listing
90 '
100 CLS:PRINT@45,"WORDSTAR TRANSMISSION PROGRAM"
110 PRINT:PRINT" Default protocol  uses the RS-232"
120 PRINT" modem at 300 baud with 8 bit,No parity,"
130 PRINT" 1 Stop  bit & Xon disabled"
140 '
150 'Optional Settings Prompt
160 '
170 PRINT:PRINT"Would you like to change the defaults? ";
180 GOSUB 990: A%=(INSTR("YyNn",I$)+1)/2
190 IF A%<>0 THEN ON A% GOTO 380,230 ELSE 170
200 '
210 'File Prompt, ".DO" Extension, & Transmission
220 '
230 CLS:INPUT"Name of file to send ";FI$
240 A%=INSTR(FI$,".DO"):IF A%<>0 THEN 260
250 FI$=FI$+".DO"
260 OPEN FI$ FOR INPUT AS 1
270 OPEN P$ FOR OUTPUT AS 2
280 C$=INPUT$(1,1)
290 PRINT #2,C$;
300 IF EOF(1) THEN PRINT #2,CHR$(26) ELSE GOTO 280
310 PRINT "DONE":CALL 21179
320 PRINT"PRESS ANY KEY TO CONTINUE":GOSUB 990:GOTO 10
330 '
340 'Options Selection
350 '
360 'Baud Rate
370 '
380 CLS
390 PRINT"Baud rate selection":PRINT
400 PRINT"0 - Modem (300)*    5 - 1200 baud"
410 PRINT"1 - 75 baud         6 - 2400 baud"
420 PRINT"2 - 110 baud        7 - 4800 baud"
430 PRINT"3 - 300 baud        8 - 9600 baud"
440 PRINT"4 - 600 baud        9 - 19200 baud"
450 PRINT"Choice ? ";:GOSUB 990
```

```
460 P1=ASC(I$)-48:IF P1<0 THEN 380 ELSE IF P1>9 THEN 380
470 P1$=I$
480 '
490 'Bit (Word) Length
500 '
510 CLS:PRINT"Word length selection"
520 PRINT:PRINT" 6 - 6 bits"
530 PRINT" 7 - 7 bits"
540 PRINT" 8 - 8 bits * "
550 PRINT"Choice":GOSUB 990
560 P2=VAL(I$):IF P2<6 THEN 510 ELSE IF P2>8 THEN 510 ELSE P2$=I$
570 '
580 'Parity
590 '
600 CLS:PRINT"Parity Selection"
610 PRINT:PRINT" I - Ignore parity"
620 PRINT" O - Odd parity"
630 PRINT" E - Even parity"
640 PRINT" N - No parity *"
650 PRINT" Choice": GOSUB 990
660 A%=(INSTR("IiOoEeNn",I$)+1)/2
670 IF A%<1 THEN 600 ELSE IF A%>4 THEN 600
680 P3$=MID$("IOEN",A%,1)
690 '
700 'Stop Bit
710 '
720 CLS:PRINT"Stop bit selection"
730 PRINT:PRINT" 1 - 1 stop bit *"
740 PRINT" 2 - 2 stop bits"
750 PRINT:PRINT"Choice ":GOSUB 990
760 A%=VAL(I$):IF A%<1 THEN 720 ELSE IF A%>2 THEN 720
770 IF A%=1 THEN P4$="1" ELSE P4$="2"
780 '
790 'X-On/X-Off Status
800 '
810 CLS:PRINT"Line status selection"
820 PRINT:PRINT" E - Enable (XON)"
830 PRINT" D - Disable (XOFF) *"
840 PRINT:PRINT"Choice ":GOSUB 990
```

```
850 A%=(INSTR("EeDd",I$)+1)/2
860 IF A%<1 THEN 810 ELSE IF A%>2 THEN 810
870 P5$=MID$("ED",A%,1)
880 IF P1$="0" THEN P$="MDM":"+P2$+P3$+P4$+P5$:CALL 21200: GOTO 600
890 P$="COM:"+P1$+P2$+P3$+P4$+P5$
900 '
910 'Selection Display
920 '
930 PRINT"Your selection was "; P$:PRINT"Is this OK? "
940 GOSUB 990:A%=(INSTR("YyNn",I$)+1)/2
950 IF A%=1 THEN 230 ELSE IF A%=2 THEN 380 ELSE 940
960 '
970 'INKEY Routine
980 '
990 I$=INKEY$:IF I$="" THEN 990 ELSE RETURN
```

How the Program Works

This program reads one character at a time from a document file and sends it out the Model 100's communications port to another computer. Unlike some other word processing software, WordStar does not use hard carriage returns at the end of each line, so this program sends the characters in the file *alone*, without the hard returns. After the program reads the last character in the file, it sends a CTRL Z which is the end-of- file marker recognized by WordStar. All of this is accomplished in lines 260 through 300. Line 260 opens a file for input; line 270 opens the RS-232 communications port; line 280 reads one character from the file; 290 writes the character out the communications port; and line 300 looks for the end of the file. If it is detected, a CTRL Z is sent to the port; if not, the program loops back for the next character in the file. These five lines are the heart of the file transfer process.

The remainder of the program is devoted to allowing the user to change the communications protocol. The program accepts valid input and places it in a string used to initialize the communications port. If the modem port is selected for modem to modem transfer, machine language calls are made: in line 880 the built-in modem "lifts" the telephone line to allow transmission, and in line 310 it disconnects the phone line after the file has been transferred.

12

Higher Math

STATS

Statistics

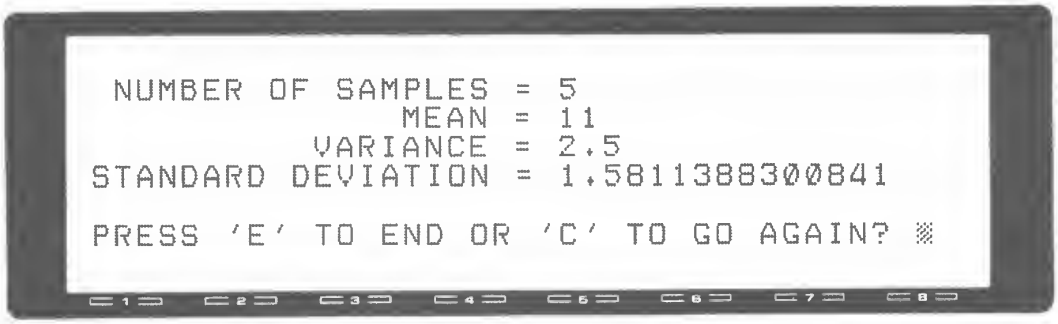
Statistical Analysis

The Space Shuttle *Columbia* has just made its sixth launch and is approximately twenty-four hours into a flawless flight. Mission Control informs *Columbia* (and those like you, listening in on NASA's open broadcasts) that its first three orbits took 7.45, 7.34, and 7.59 hours, respectively. You use your Model 100 and this Statistics program to enter the data from Mission Control. In seconds the Model 100 display shows that mean orbital time is 7.46 hours; the variance, .0157 hours; and the standard deviation, .12529964086141 hours. These figures are essential to Mission Control in plotting *Columbia's* orbit; in fact, one of *Columbia's* on-board computers has a similar program.

"Great!! The standard orbital deviation is negligible!" Flight Commander MacAliffe reports. "The stabilizing retro-rocket must be working like a Swiss watch."

"A very expensive Swiss watch!" Mission Control reminds him.

You can use your new "Stat" program to derive the mean, variance, and standard deviation of any group of numbers. These figures are important to scientists and engineers who must draw conclusions from a large amount of experimental data. However, anyone can apply them to a variety of everyday situations, such as analyzing sales figures or meteorological data.



In statistics the mean is the average of all the data; the variance is a measure of the "departure" of the data from the mean; and the standard deviation is the square root of the variance. A large standard deviation indicates a data group with little cohesion. On the other hand, a small standard deviation evinces greater cohesion, as in our shuttle example where the orbits were quite similar in length. Though standard deviation is the most common way to express the degree of departure from the mean, variance is sometimes preferred to demonstrate this departure.

In order to compute these statistics for a data group, type in the numbers from your data set and then enter "E" to end the set. The statistical values derived from your entries will then be displayed. To enter a new set of entries, press "C" and the program variables will be reset.

Program Listing

```
10 '
20 '  STATS
30 '
40 'Intialization of Variables & Start Screen
50 '
60 ME=0:N=0:B=0
70 CLS:PRINT@80,"ENTER DATA or 'E'nd:  #";N+1;
80 INPUT D$
90 L$= LEFT$(D$,1)
100 IF L$="E" OR L$="e" THEN 240
110 A=ASC(L$)
120 IF A<48 THEN 420 ELSE IF A>57 THEN 420
130 '
140 'Values Stored
150 '
160 V=VAL(D$)
170 N=N+1
180 ME=ME+V
190 B=B+V*V
200 GOTO 70
210 '
220 'Statistical Calculations
230 '
240 CLS:ME=ME/N
250 VA=(B-N*ME*ME)/(N-1)
260 SD=SQR(VA)
270 '
280 'Values Displayed
290 '
300 PRINT:PRINT" NUMBER OF SAMPLES =";N
310 PRINT"              MEAN =";ME
320 PRINT"          VARIANCE =";VA
330 PRINT"STANDARD DEVIATION =";SD
340 PRINT:INPUT"PRESS 'E' TO END OR 'C' TO GO AGAIN";F$
350 '
360 'Restart/Continue Loop
370 '
380 IF F$="E" OR F$="e" THEN STOP ELSE
    IF F$="C" OR F$="c" THEN 60 ELSE
    CLS:PRINT"PLEASE TRY AGAIN":GOTO 340
390 '
400 'Error Trapping
410 '
420 CLS:PRINT@80,"BAD DATA, RE-ENTER:   #";N+1;
430 GOTO 80
```

How the Program Works

This program operates by taking the values you enter, storing them as the variables ME and B in lines 180 and 190, and then performing the fundamental statistical calculations in lines 240 through 260. The mean is calculated in 240 simply by dividing the total of the entered data by the number of entries. The variance in 250 is a function of the mean squared and the sum of squares of each entry. The standard deviation in 260 is the square root of the variance.

The initialization of the variables and the initial entry screen are set up in lines 60 to 80. Lines 300 through 340 display the values for the statistical computations. The program will trap non-numeric entry errors in lines 120 and 420. The restart/continue loop in 380 allows you to enter another set of data without rerunning the program, or lets you end the program without pressing (SHIFT) (BREAK).

A string variable is used for the input variable because an alpha character, E, may then be used as a "control" to begin the calculation routine. The string must be converted back to a numeric value. These extra lines for the string conversions are necessary; otherwise, a numeric value, such as −99, would have to be employed as an entry "end flag", and thereby be eliminated as a possible numeric entry.

PROBS

Probability

Chance of Certain Events

You have just opened the sixth bottle of champagne and are beginning to wonder if the "computer warming" party for your new Model 100 was a good idea, when one of your guests poses a question: what are the chances that two people in the room have the same birthday? There are about two dozen guests, so you figure the odds are slight. Should you bet on it?

A festive voice suggests the Model 100 be consulted. Eager to display the computer's talents, you load your Probability program. The guests gather around. You type in 365 (the number of possible birthdates, excluding leap year babies) in response to the prompt. One of the guests reports that the number of people in the room totals 23. The odds scroll by, then you press a key to stop the scroll at 23. Your friends hush . . . and the Model 100 reveals that the probability of two people at the party having the same birthday is better than 50 percent! 52.1, to be exact.

Probabilities can be surprising — it's a good thing you didn't bet!

The computation of probability depends on the theorem that the probability of a single occurrence happening can be calculated by dividing the total number of possible occurrences by the total number of possibilities. In the example above, the program actually computed the probability that no two guests had the same birthday and then subtracted that "reverse"

```
 6                .627
 7                .776
 8                .888
 9                .957
 10               .989
 11               .999

PRESS ANY KEY TO START OR STOP
```

probability from one. In other words, the program starts with one partygoer (the probability being zero since there is only one person), adds another, and determines that the probability of these two *not* having the same birthday is 364/365. The probability that a third celebrant does *not* have the same birthday as the first two is 363/365. This pattern continues until the number of "events" (the total number of people being considered) is reached. Finally, the program calculates the probability of two people sharing the same birthday, so the formula is one minus the probability that everyone has a different birthday or:

1-((364/365)*(363/365)*(362/365). . .)

If we had to use a pencil and paper or even a pocket calculator, it would be quite a task to arrive at our 52.1 percent figure. But with the Model 100 and this Probability program these kinds of calculations are a snap. Just answer the "number of units" prompt with the total number of possibilities and press ENTER. You can press any key on your Model 100 to start and stop the display in order to examine the results. Once all the probabilities have been displayed, you can rerun the program by pressing ENTER. To get back to BASIC, press SHIFT BREAK.

Program Listing

```
10 '
20 '  PROBS
30 '
40 'Start Screen & Prompt Lock
50 '
60 CLS
70 INPUT"NUMBER OF UNITS IN SAMPLE ";NU
80 PRINT@280,"PRESS ANY KEY TO START OR STOP";:CALL 16949:PRINT
90 '
100 'Probability Calculations Routine
110 '
120 Q=1:NO=NU
130 FOR N=1 TO NO
140 X$=INKEY$:IF X$<>"" THEN GOSUB 260
150 Q1=INT((1-Q)*1000+.005)/1000:IF Q1>1 THEN Q1=1
160 PRINT N,Q1
170 Q=Q*(NU-N)/NU:NU=NU-1
180 NEXT N
190 '
200 'Unlock Prompt Line & Continue
210 '
220 INPUT"PRESS ENTER TO CONTINUE";X$
230 CALL 16954:GOTO 20
240 '
250 'Pause Routine
260 '
270 X$=INKEY$:IF X$="" THEN 260 ELSE RETURN
```

How the Program Works

The calculations routine in lines 120 through 180 forms the base for the program's computations. Lines 170 and 180 contain the core equation where the value of N increases by one with each loop of the subroutine. (NU-N) represents the decreasing number of possibilities as the program loops. As the possibilities decrease, the probability (represented by the variable Q1) increases. Line 150 limits the displayed probability to three decimal places and also makes certain that it will not exceed 100 percent.

Lines 80 and 230 use two assembly language subroutines in an interesting manner. Because the program scrolls rapidly, a routine to start and stop the display has to be employed. However, the start-stop prompt would scroll along with the results in a very annoying way without the clever use of a call normally engaged to lock the last line of the display when the LABEL key is pressed. In line 80 the start-stop prompt is displayed as the last line, then *locked* as if it were the label prompt. Before the program is finished (line 230), the prompt line is unlocked, and you may start again.

You need not be concerned if you use the BREAK key to stop the program and the start-stop prompt remains. You can use the LABEL key to turn it off, or it will automatically shut down when you return to the menu. These assembly language routines, built into the Model 100, can be employed whenever you need to lock a displayed line at the bottom of the screen.

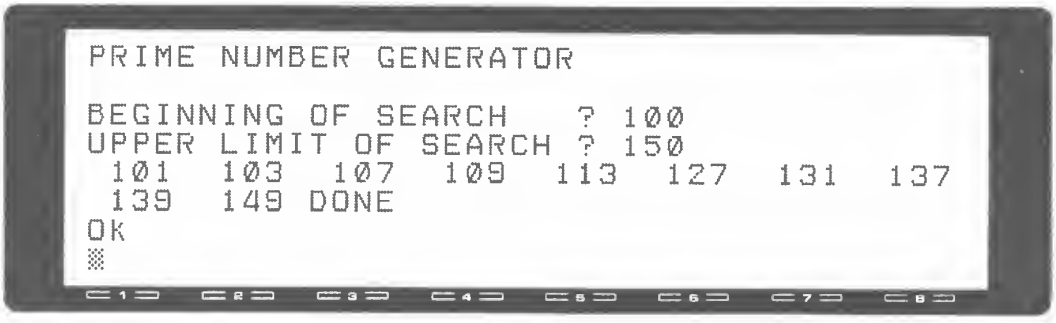
Primes

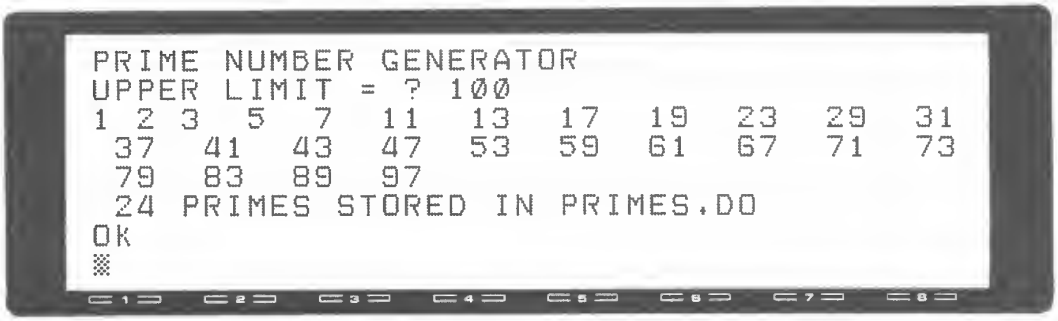
Two Ways to Generate Prime Numbers

What is a prime number? Why generate them? A prime number is any number that is divisible without remainder *only* by itself and one. They have a variety of uses in mathematical theory and provide excellent test data for programs that manipulate large numerical sets. Prime numbers are the building blocks of mathematics, since every integer can be constructed of prime numbers. Thus, 42 can be reduced to 2 times 3 times 7, which are all primes.

Many important questions in mathematics concern prime numbers. One of the strangest aspects of prime numbers is that they don't seem to follow any predictable pattern. Try using one of these prime number programs to display some primes. Can you find a pattern? No one else can either, including some of the most brilliant minds in mathematics. If you find a rule that can predict which numbers will be prime, you will assure yourself a place in history.

Other unknown questions about primes remain to be solved. For instance, some primes come in pairs, like 17 and 19. That is, the difference between such primes is exactly 2. The question, as yet unanswered, is whether there are an infinite number of such prime pairs, or whether, past a certain point, all the primes become so far apart that no more prime pairs





occur. If you can prove this one way or the other (perhaps with the use of these programs), you will again assure yourself a place in the mathematics hall of fame.

The first few prime numbers are easy to spot: 1, 2, 3, 5 .... (Two is the only even prime number; every other even number is divisible by two as well as by itself, and therefore is not prime). But how about determining the prime numbers between 250 and 300? It would be difficult to find them without your Model 100 and these Prime Number programs.

The first program (PRIME1) takes a traditional approach to determine if numbers are prime. Each number is taken individually and divided by every number up to itself. If divisible by any number other than itself or one, the number under scrutiny is not prime. This approach has the advantage of being simple, but, unfortunately, it is extremely time-consuming when used on large numbers.

When this program is run, you'll be prompted for "beginning of search" and "upper limit of search". If the difference between the upper limit and the beginning is over 200, the program displays a message to tell you that it will take a few minutes to run. In fact, you'll have to let this program run *overnight* for sets with a large limit difference!

The second program (PRIME2) uses a different approach. It sets flags in an array to indicate that a particular division has been tried, and thus avoids the repetitive checking that slows the first program as the numbers get larger and larger. When you run this second program you'll be asked to enter the upper limit only. The program calculates all the prime numbers from one to the limit value you enter.

When PRIME2 is run, you'll find that the numbers are generated faster and faster as the prime numbers get larger, rather than the reverse. The trade-off for this speed is that your upper limit choice is limited by the amount of free memory that your Model 100 has available. Since each number is put into an array in resident memory, and larger numbers take up more memory than smaller ones, eventually, very large values will ex-

haust your Model 100's free memory, and you'll receive an "OM" (out of memory) error. However, the efficient calculating method employed by this program makes it more desirable than the slower PRIME1 for most situations where you need to determine primes.

Both prime generator programs record the prime numbers they find in a document file named "PRIMES.DO". This allows you to scroll through the primes or use the ones listed, after the program has run. If you have a printer available, you may want to modify these programs so that the output is listed on the printer rather than being sent to a file.

Program Listings

```
10 '
20 '  PRIME1
30 '
40 ' Open "PRIMES.DO" for Output
50 '
60 MAXFILES = 1:OPEN "PRIMES.DO" FOR OUTPUT AS 1
70 '
80 'Initial Screen and Input Queries
90 '
100 CLS:PRINT"PRIME NUMBER GENERATOR":PRINT
110 INPUT"BEGINNING OF SEARCH   ";B
120 INPUT"UPPER LIMIT OF SEARCH ";L
130 '
140 'Pause Message
150 '
160 IF L-B>200 THEN PRINT"This is going to take me a few minutes"
170 '
180 'Prime Search Loop
190 '
200 FOR X=2 TO INT(B/2)
210 Y=B/X
220 IF Y-INT(Y)=0 THEN 250
230 NEXT X
240 PRINT B;:PRINT#1,B;
250 B=B+1:IF B>L THEN CLOSE:PRINT "DONE":END
260 GOTO 200
```

```
10 '
20 '  PRIME2
30 '
40 'File and String Definition
50 '
60 MAXFILES =1
70 DEFINT A-Z:DIM A(1000)
80 '
90 'Initial Screen, Input Query, & Output File Opened
100 '
110 CLS:PRINT"PRIME NUMBER GENERATOR"
120 INPUT "UPPER LIMIT = ";U
130 PRINT"1 2";:OPEN "PRIMES.DO" FOR OUTPUT AS 1
140 '
```

```
150 'Prime Search Loop
160 '
170 FOR X=0 TO U
180 IF A(X)=1 THEN 250
190 Y=X+X+3:IF Y>U THEN 300
200 PRINT Y;:PRINT #1,Y;
210 K=X+Y
220 IF K>U THEN 240
230 A(K)=1:K=K+Y:GOTO 220
240 C=C+1
250 NEXT X
260 '
270 ' Close Output File & Display Number of Primes
280 '
290 CLOSE
300 PRINT:PRINT C;"PRIMES STORED IN PRIMES.DO"
```

How the Programs Work

Prime Number Generator #1 operates on a "grind away at it" principal. It takes all of the numbers between the lower and upper limits specified and checks individually to see if they are divisible by any number other than one (lines 210 and 220). In line 200 you'll notice that the program only loops "FOR X=2 TO INT(B/2)"; in other words, up to half of the numbers in a value. This saves run time. It works because a number not divisible by a value, when half of that particular number is reached, will not be evenly divisible by a number *more* than half its value.

Line 250 adds one to the previous value and checks to see if the new value is greater than the upper limit. If it is, then the output file is closed and a "DONE" message is displayed.

The second Prime Number Generator is considerably different from the first. In order to save run time, all of the numbers up to the limit value entered are stored in an array. In the program's progressive search for prime numbers, each value found that is *not* prime is flagged (line 180) and eliminated from future computations. In this way, the program bypasses repetitious calculations that slow down the first program. This second prime number generator starts off slowly but picks up speed as more and more numbers are flagged and eliminated.

Again, because this program depends on having each number in an array, you must be careful not to exceed the available memory in your Model 100. The program is initially set with an upper limit of 1000 ("DIM A(1000) in line 70). You may increase or decrease this number, depending upon the amount of memory your Model 100 has free.

Line 240 acts as a counter for the number of primes written to the PRIMES.DO file. The total is displayed (line 300) after the entire range of numbers has been read and the output file closed (line 290).

13

File Anatomy and File Length

LISTFI

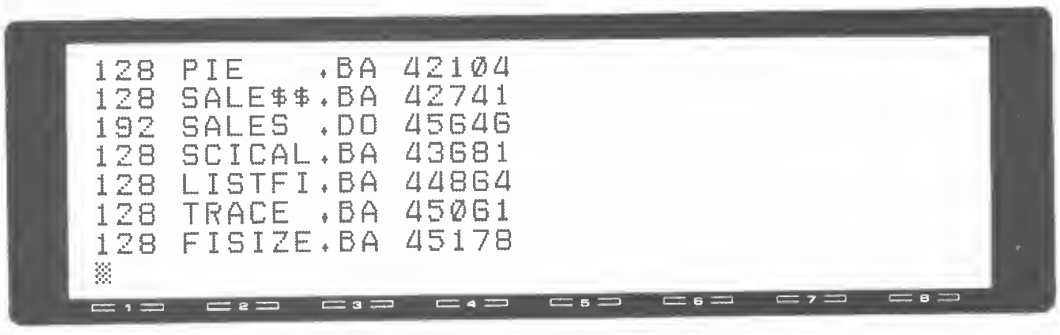
Listfile

Inside the Model 100 File Directory

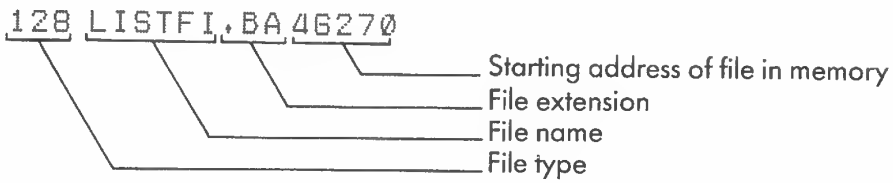
This program and the one following permit you to sample the forbidden world within the Model 100. They give you a view of what goes on beneath the "user-friendly" surface of the computer. This program can show you, among other things, how to find where other programs are actually located in the Model 100's memory. The next program, Trace, will then show you how to dissect a BASIC program (or any other file) to find out what really makes it tick.

Listfile is a BASIC program which examines the contents of the Model 100's built-in file directory. This is a special area in the computer's memory where all information about files currently in memory is stored. Each time you type in a BASIC program or create a text file, and store it in memory, its name is placed in the file directory. The directory can hold 19 files, corresponding to the 19 free locations on the main menu.

There are no parameters to enter when you run this program. When you start it, the name of each file in the directory is printed out, along with



the file type and its starting address in memory. A typical output line for one of the 19 files would look like this:



The starting address of the file is the actual memory address where the program or text file begins. This is a very useful thing to know, as we will see in the next program.

The "file type" is a one-byte number which tells the operating system what kind of file it is. For instance, if the file is a BASIC file with the extension .BA, the file type will be 128. If it is a text file with the extension .DO, then its type will be 192. An interesting file type is the number 0, which you may see in some of your files. This means that the file has been erased with "KILL". When a file is KILLed, it is not immediately blotted out of memory. Instead, its file type is set to 0 to notify the operating system that the memory this file occupies and its entry in the file directory are free to be used again. Only when you enter new files are these spaces actually written over.

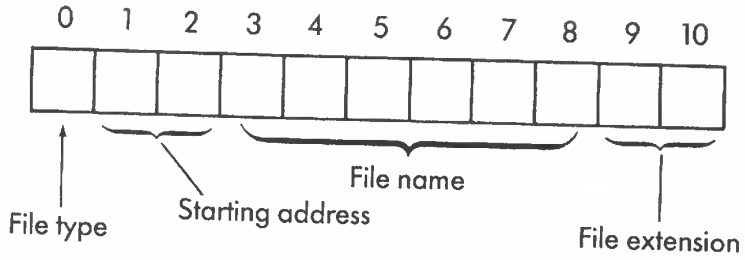
When you run this program, use the PRINT key so that you get a permanent record of all the file addresses on your printer. Save this printout to use with the next program.

Program Listing

```
100 '  LISTFI
110 '
120 FOR FILE=0 TO 18
130 PTR=FILE*11+63930
140 '
150 'Print file type
160 '
170 PRINT USING "### "; PEEK(PTR);
180 '
190 'Print file name
200 '
210 FOR J=3 TO 8
220 PRINT CHR$(PEEK(PTR+J));
230 NEXT J
240 PRINT",";
250 '
260 'Print file extension
270 '
280 FOR J=9 TO 10
290 PRINT CHR$(PEEK(PTR+J));
300 NEXT J
310 '
320 'Print starting address
330 '
340 PRINT PEEK(PTR+1)+PEEK(PTR+2)*256
350 NEXT FILE
```

How the Program Works

The file directory is located in memory starting at address 63930 (decimal). Each entry in the directory contains 11 bytes, arranged like this:



The program's job is to display the information contained in these directory entries in a form that looks comprehensible on the screen.

The entire program is nestled in a FOR...NEXT loop which counts through all 19 files, from 0 to 18. The first time through this loop the variable FILE is 0, so the entry that starts at 63930 will be examined as calculated in line 130. The next time through, FILE will be 1, so 11 will be added to 63930, and so on. In line 130 the variable PTR is set to the start of each directory entry.

In line 170 the program prints the file type. It finds this by PEEKing into the location at PTR.

The file name is printed in lines 210 to 230, using a FOR...NEXT loop and the variable J to step through the locations from PTR + 3 to PTR + 8.

In a similar way, lines 280 to 300 print the file extension from locations PTR + 9 and PTR + 10.

Finally, the starting address of the file is printed in line 340. Since this address occupies two bytes, each byte must be PEEKed into separately, and the results combined to yield the final address. The low-order (least significant) part of this number is in PTR + 1, and the high-order part is in PTR + 2. A one-byte number can only count up to 255 (decimal). Accordingly, PTR + 2 must be multiplied by 256 and added to PTR + 1 to calculate the final number.

When the information has been printed out for all 19 files, the program ends.

TRACE

Trace

Model 100 Programs and Files

This program acts as a software X-ray machine, permitting you to see what's going on in the Model 100's memory. It is very useful if you are interested in the internal structure of BASIC programs. Understanding how BASIC programs are constructed is essential if you want to write certain kinds of programs, such as line renumberers and BASIC editing or debugging programs.

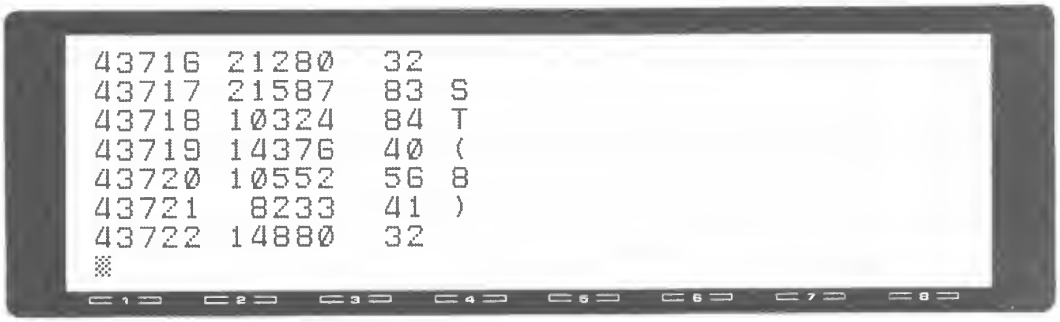
This program is easy to run. When you start it, you will be asked the address you want to trace from. Respond by typing in the address you found by using the Listfile program described earlier. This is where the trace will begin.

Let's create a short program to experiment on. We'll call it HELLO. Here it is:

```
10 PRINT"Hello"
20 END
```

Let's say we used the Listfile program to find the starting address of HELLO. The output when we run Listfile might look like this:

```
128 HELLO .BA 33090
```



This tells us that the HELLO program starts at memory location 33090. Now we run the trace program and answer "33090" to the question, "Trace from?" (Of course, this address will be different on your computer, since you may have a different amount of memory and different programs saved.)

Trace looks at the individual bytes in the BASIC program. It provides four output columns for each byte. The first is the memory location of the byte. The second is the two-byte number formed from the current byte and the one following. The third is the one-byte contents of the current byte, and the fourth is the ASCII character, if any, which this number represents.

Here's the output that results when TRACE is applied to our sample HELLO program.

```
33090 33103   79 O      ← Pointer to next line at 33103 (high byte of pointer)
33091  2689  129 X
33092    10   10        ← Line number 10 (high byte of number is 0)
33093 41728    0
33094  8867  163 X      ← Keyword for PRINT
33095 26658   34 "      ← Quotes start message
33096 25960  104 h      ← Start of 'hello'
33097 27749  101 e
33098 27756  108 l
33099 28524  108 l
33100  8815  111 o
33101    34   34 "      ← Quotes end message
33102 21760    0        ← 0 ends program line
33103 33109   85 U      ← Pointer to next line at 33109 (high byte of pointer)
33104  5249  129 X
33105    20   20        ← Line number (high half is 0)
33106 32768    0
33107   128  128 X      ← Keyword for END
33108     0    0        ← 0 ends program line
33109     0    0        ← Two 0s mark end of program
33110     0    0
```

Now you can see what's really going on in a BASIC program. The first two bytes in each line form an address. This address is a pointer to the *next* line in the program. Thus, at address 33090 we find the address 33103, the address of the next BASIC program line (line 20). This linkage makes it easy for BASIC to search through a program looking for a certain line number.

The next two bytes form the BASIC line number. In our example this is 10, so the high half of the number is 0.

Next, we find the code for the BASIC keyword PRINT. This is a single byte with a value of 163. (To save space in the program, all BASIC keywords are coded as single bytes.)

The quotes and the material inside them are stored as simple ASCII characters. The program line ends with a single 0 byte.

With this introduction you should be able to explore the hidden world behind your BASIC programs. Try Trace on more complex programs. You may be amazed at what you discover.

Program Listing

```
100 '  TRACE
110 '
120 INPUT"trace from";J
130 XC=PEEK(J)+PEEK(J+1)*256
140 '
150 'Print address, contents of address: two bytes, one byte
160 '
170 PRINT USING"##### ##### ###";J;XC;PEEK(J);
180 '
190 'Print ASCII symbol
200 '
210 PRINT " ";CHR$(PEEK(J))
220 J=J+1 : GOTO 130
```

How the Program Works

As you can see, this is actually a very simple program. The user inputs a value for the variable J, which is the address to be traced. XC is the two-byte address formed from the contents of the current byte and the following byte.

Line 170 prints out J, XC, and the one-byte contents of J. The ASCII value of this byte is printed out in line 210.

J is then stepped to the next memory byte, in line 220, and the program goes back to print the information for this new byte.

FISIZE

File Length

Lister Shows Size of All Files on Menu

Have you ever gotten the dreaded OM (out of memory) error message? Are you curious how long your BASIC programs or text files are? On large computers with disk operating systems there is usually a command which lists all the files in the system and shows the length in bytes of each one. Unfortunately, there is no such command in the Model 100, since the files are simply displayed in the main menu.

FISIZE fills this gap — when you run this program, it displays the name and length of each file.

Once the program has been typed in, operating it is very simple. From the main menu, position the cursor over the program name, FISIZE, and press ENTER. The file names and their sizes will scroll by. If you want to stop the display, press PAUSE. Press PAUSE again to continue.

Program Listing

```
100 '  FISIZE
110 '
120 LAST=PEEK(64433+1)+PEEK(64433+2)*256
130 '
140 FOR FILE=0 TO 18
150 PTR=FILE*11+63930
160 IF PEEK(PTR)=0 THEN 510
170 '
180 'print filename
190 '
200 FOR J=3 TO 8
210 PRINT CHR$(PEEK(PTR+J));
220 NEXT J
230 PRINT",";
240 '
250 'print extension
260 '
270 FOR J=9 TO 10
280 PRINT CHR$(PEEK(PTR+J));
290 NEXT J
300 '
310 'Calculate length
320 '
330 START=PEEK(PTR+1)+PEEK(PTR+2)*256
340 TLAST=LAST
350 '
360 'search for next largest address
370 '
380 FOR SUBFILE=0 TO 18
390 PT2=SUBFILE*11+63930
400 IF PEEK(PT2)=0 THEN 450
410 ADDR=PEEK(PT2+1)+PEEK(PT2+2)*256
420 IF ADDR<=START THEN 450
430 IF ADDR>TLAST THEN 450
440 TLAST=ADDR
450 NEXT SUBFILE
460 '
470 'print length
480 '
490 LNGTH=TLAST-START
500 PRINT USING"#####";LNGTH
510 NEXT FILE
```

How the Program Works

If you've read the description of LISTFI earlier in this section, you'll understand how the file directory is structured. In fact, the first part of FISIZE, down to line 290, is very much like the LISTFI program.

The new part of this program calculates the length of the files. The only information contained in the file directory is the *starting address* of each program. The lengths of the programs must be calculated by taking the starting address of a program, then looking for the next highest starting address. The assumption here is that each program stops where the next program begins, an assumption that seems to correspond to the way the Model 100 stores files.

One problem arises using this scheme: What happens to the last program? There's no program following it in memory, so we don't have a starting address for the next program to tell us the ending address of this program. Fortunately, there is a place in memory where this address is stored — a "dummy" file at 64433. So, at the beginning of the program (line 120), we find out this "end-of-all-files" address and assign it to the variable LAST.

FISIZE goes through all the files, one by one, using the loop from 140 to 510. In the lines down to 290 the filename and extension are printed out. The starting address of the file is then obtained in line 330 and assigned to the variable START.

The strategy now is to go through all the files again, looking for the next-largest starting address. A variable TLAST starts off being set to the "end-of-all-files" address, LAST. If a file has a starting address less than this, TLAST is changed to this new address. Thus, when all files have been examined, the variable should contain the address of the next-largest file.

Let's look at this process in more detail. First, each file is checked (line 400) to make sure its file type is not 0, indicating it has been erased. Erased files are skipped, and the program goes to the NEXT in line 450 to get the next file. If it's a valid file, its starting address ADDR is obtained in line 410. The address is checked in line 420 to see if it's less than START. If so, it can't be the next-largest file, so it's skipped. Then in line 430 ADDR is compared with TLAST. If it's greater, the next file is obtained. If not, then TLAST is updated to reflect the new value of ADDR.

This process is continued through all the files, so finally TLAST must contain the next-largest address. The difference between the START address and TLAST is, then, the length of the file. This is calculated in line 490 and printed out in line 500.

It would have been possible to speed up the operation of this program by placing the files with their starting addresses in an array, and sorting the array. Once the files were sorted by starting address, it would be easy to print out their lengths. However, this involves a somewhat longer and more complex process than the program shown here. The current program is both shorter and easier to understand than one using the sorting principle.

Index

(0452)

Other PLUME/WAITE books on the TRS-80® Model 100:

☐ **Introducing the TRS-80® Model 100, by Diane Burns and Sharyn Venit.** This book, intended for newcomers to the Model 100, offers simple step-by-step explanations of how to set up your Model 100 and how to use its built-in programs: TEXT, ADDRSS, SCHEDL, TELCOM, and BASIC. Specific instructions are given for connecting the Model 100 to the cassette recorder, other computers, the telephone lines, the optional disk drive/video interface, and the optional bar code reader. (255740—$15.95)

☐ **Mastering BASIC on the TRS-80® Model 100, by Bernd Enders.** An exceptionally easy-to-follow introduction to the built-in programming language on the Model 100. Also serves as a comprehensive reference guide for the advanced user. Includes all Model 100 BASIC features including graphics, sound, and file-handling. With this book and the Model 100 you can learn BASIC anywhere! (255759—$19.95)

☐ **Practical Finance on the TRS-80® Model 100, by S.D. Venit and D.K. Burns.** The perfect book for anyone using the Model 100 in business: investors, real estate brokers, managers. Contains short but powerful programs to perform production planning, and access financial and other information from CompuServe® and the Dow Jones News/Retrieval® service. (255767—$15.95)

☐ **Hidden Powers of the TRS-80® Model 100, by Christopher L. Morgan.** This amazing book takes you deep inside the Model 100 to reveal for the first time how it really works. You'll learn about the amazing power buried in the ROM, and how to use this power in your own programs. You can print in reverse video, prevent any screen lines from scrolling, dial the telephone from BASIC, control external devices from the cassette port, and discover many other fascinating secrets hidden within your Model 100. (255783—$19.95)

All prices higher in Canada.

To order, use the convenient coupon on the next page.

The Waite Group

COMPUTER • Z5577 • $16.95
CANADA • $21.95

PLUME
WAITE

GAMES & UTILITIES FOR THE TRS-80® MODEL 100

The action-packed graphic games and practical software tools in this guide are fun to learn, and reveal powerful programming techniques for your Model 100. Draw pictures, learn to type, play music, manipulate liquid crystal pixels, list file lengths, transfer files to larger computers, or turn your Model 100 into an advanced calculator.

This book contains well-designed programs for engineers, musicians, businesspeople, designers, students, and anyone who wishes to utilize the power of the Model 100 fully. Programs cover two kinds of calculators, sorting, math, statistics, Morse code, word counting, games of all types, and exploring the ROM. You'll have a great time playing INTERJERK INVADERS, DUELING DIGITS, or TYPOMANIA!

The operation of each program is explained in detail, and the program listing is fully annotated, so you can customize it as you wish, or study its operation to improve your own programming.

The Waite Group is a Sausalito, California, based producer of high-quality books on personal computing. Acknowledged as a leader in the industry, the Waite Group has written and produced over thirty titles, including such best sellers as Assembly Language Primer for the IBM PC & XT, Graphics Primer for the IBM PC, CP/M Primer, and Soul of CP/M. Internationally known and award winning, Waite Group books are distributed worldwide, and have been repackaged with the products of such major companies as Epson, Wang, Xerox, Tandy Radio-Shack, NCR and Exxon. Mr. Waite, President of the Waite Group, has been involved in the computer industry since 1976 when he bought his first Apple I computer from Steven Jobs.

25577
0 51857 01695
ISBN 0-452-25577-5